VERSIONE PLUS ✔ RIVISTA+LIBRO+CD €8,90
VERSIONE STANDARD RIVISTA+CD €6,90
VERSIONE LIGHT RIVISTA €3,90
Periodicità mensile • OTTOBRE 2003 • ANNO VII, N.9 (73)
Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL

ioProgrammo

CRITTOGRAFIA FACILE
CHIAVE PUBBLICA E CHIAVE PRIVATA

Reverse Engineering

CRACK SI FA COSÌ

Scopri come sproteggere le applicazioni e come difendere il tuo codice

Strumenti
Tecniche
Esempi pratici

REPORTAGE ESCLUSIVO Tech·Ed 2003

SISTEMA
- Non si butta niente: utilizzare i componenti COM in .NET
- Prestazioni in Java: fai correre le tue applicazioni
- Utilizzare caselle di riepilogo e caselle combinate in .NET

INTERNET
- Invio e ricezione di E-mail in Visual Basic

PALMARI
- XML mobile: creare pagine per i cellulari con XHTML-MP
- GPS e Pocket PC: rendering e gestione centralizzata delle mappe

ELETTRONICA
- I sensori del tatto nella costruzione di un robot
- Introduzione alla progettazione in LabView

CORSI
- Visual Basic, Java, C++, VB.NET, C#, Matlab

Metti il turbo a Visual Basic

Rinnova l'interfaccia con funzionalità inedite e nuovi effetti speciali

JAVA IN PRATICA
- Generare un file Excel
- Disegnare grafici statistici

DELPHI SENZA FRONTIERE
Localizza ora: i tuoi software in tutte le lingue!

ISSN 1128-594X
EDIZIONI MASTER
www.edmaster.it
ioProgrammo (plus) Anno VII - N° 9 (73) • € 8,90

TESTATO
SoftICE Driver Suite
Il miglior debugger per Windows

CINEMA 4D: EFFETTI DIGITALI ALLA PORTATA DI TUTTI

Contents

Anno VII - n. 9 (73) Ottobre 2003

Cronaca di un successo annunciato

ioProgrammo.it è partito e, anche se è forse presto per dirlo, non sarei onesto se non manifestassi tutta la soddisfazione della redazione e mia personale per l'eccezionale successo che ha riscontrato. Centinaia di iscritti al forum già nelle prime settimane di vita ed una partecipazione superiore ad ogni aspettativa.

Molti degli autori di ioProgrammo sono stati coinvolti nell'impresa e la comunità che si sta raccogliendo attorno al sito sembra davvero entusiasta, oltre che numerosa.

Pressoché tutti i quesiti tecnici posti sul forum hanno trovato una risposta in poche ore, sia grazie agli autori della rivista sia per l'adesione spontanea di molti lettori che hanno trovato in **ioProgrammo.it** una nuova casa. Inutile nascondere che anche per noi della redazione il sito si sta rivelando di grande utilità: l'immediato feedback ai contenuti della rivista si è dimostrato prezioso e, per il futuro, non dubito che qualche nuovo collaboratore sarà scelto fra i partecipanti più attivi del forum.

L'aspetto più avvincente è che il sito si è dimostrato un organismo in continua evoluzione: giorno per giorno, grazie ai vostri suggerimenti, abbiamo aggiunto o modificato numerose sezioni ed il sito è ancora in crescita. Allo studio ci sono moltissime nuove iniziative ed in particolare stiamo lavorando ad un legame ancora più stretto fra la rivista ed il sito: restate sintonizzati e, se avete qualche idea, non esitate a proporla!

raffaele@edmaster.it

Da questo numero, all'inizio di ogni articolo, troverete un nuovo simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *soft**codice*\ e *soft**tools*\) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it. Per scaricare software e codice da Internet, ogni mese indicheremo una password differente. Per il numero che avete fra le mani la combinazione è:

Username: ***norad*** *Password:* ***joshua***

ioProgrammo

Anno VII - N.ro 9 (73) - Ottobre 2003 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.ioprogrammo.it

Direttore Editoriale Massimo Sesti
Direttore Responsabile Romina Sesti
Responsabile Marketing Antonio Meduri
Responsabile Editoriale Gianmarco Bruni
Editor Gianfranco Forlino
Coordinamento redazionale Raffaele del Monaco
Redazione Antonio Pasqua, Thomas Zaffino
Collaboratori A. Aiello, M. Autiero, M. Bigatti, L. Buono, M. Del Gobbo, E. Florio, F. Grimaldi, A. Ingegneri, A. Lippo, F. Lippo, A. Marroccelli, S. Meschini, D. Pascuzzi, A. Pelleriti, C. Pelliccia, P. Perrotta, F. Sara, L. Spuntoni, E. Tavolaro, F. Vaccaro, M. Valeri, V. Vessia, D. Visicchio.
Segreteria di Redazione Veronica Longo

REALIZZAZIONE GRAFICA CROMATIKA S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

REALIZZAZIONE MULTIMEDIALE SET S.r.l.
Coordinamento Tecnico Piero Mannelli
Ideazione Grafica Gianluca Carbone
Realizzazione CD-Rom Paolo Iacona

PUBBLICITÀ Master Advertising s.r.l.
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 8321612 - Fax 02 8321754
e-mail advertising@edmaster.it
Rete Vendita Serenella Scarpa, Cornelio Morari, Roberto Piano
Segreteria Ufficio Vendite Daisy Zonato

EDITORE Edizioni Master S.r.l.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 8321482 - Fax 02 8321699
Sede di Rende: C.da Lecco, zona industriale - 87030 Rende (CS)
Amministratore Unico: Massimo Sesti

ABBONAMENTO E ARRETRATI
ITALIA: Abbonamento Annuale: ioProgrammo Basic(11 numeri): € 52,90 sconto 30% sul prezzo di copertina € 75,90. ioProgrammo Plus (11 numeri + 6 libri): € 86,90 sconto 30% sul prezzo di copertina € 123,90.
ESTERO: Abbonamento Annuale: ioProgrammo Basic (11 numeri): € 151,80. ioProgrammo Plus (11 numeri + 6 libri): € 257,00
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 028321482.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 028321699, oppure via posta a sEDIZIONI MASTER via Cesare Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: Inviare il CD-Rom difettoso in busta chiusa a: Edizioni Master Servizio Clienti - Via Cesari Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
tel.02 8321482
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelleria, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Delux Italy S.r.l. - via Rossini, 4 - Tribiano (MI)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Settembre 2003

Edizioni Master edita:
Idea Web, Go!OnLine Internet Magazine, Win Magazine, PC Fun extreme, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, <tag/>, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, La mia videoteca, Le Collection.

LA PENNA DI APACHE SU J2EE

Geronimo è il nome in codice di un nuovo progetto del gruppo Apache, rivolto alla realizzazione di una piattaforma J2EE open source che avrà licenza Apache. Il progetto è ancora ad uno stato primitivo ed è aperta a chiunque vorrà contribuire.
Lo scopo è riuscire ad avere una piattaforma certificata J2EE compliant e pare che anche Sun Microsystem guardi con favore questa iniziativa.

http://incubator.apache.org/projects/geronimo.html

UN SUPER-COMPUTER IN UN CHIP

IBM ha trovato un alleato nella realizzazione di un chip in grado di effettuare oltre mille miliardi di operazioni al secondo, ovvero più di molti supercomputer attualmente in uso.
L'Università del Texas collaborerà al progetto fornendo l'architettura TRIPS (Tera-op Reliable Intelligently adaptive Processing System).
Alla base della nuova architettura ci sarebbe un nuovo concetto detto "Esecuzione block-oriented": a differenza degli attuali chip, capaci di effettuare solo poche istruzioni contemporaneamente, la nuova architettura consentirebbe di eseguire grossi blocchi di operazioni simultaneamente. Il compimento del progetto è previsto per il 2010, ma già nei prossimi tre anni sono previsti i primi prototipi funzionanti.

www.ibm.com

News

MOLTO RUMORE PER GOOGLE

Da un anno gira voce che Microsoft voglia entrare in forze nel settore dei motori di ricerca. L'obiettivo sarebbe, ovviamente, quello di scalzare sua maestà Google dal suo stabile trono di miglior motore di ricerca esistente. Negli ultimi mesi, l'apparizione di alcuni crawler targati Microsoft aveva dato implicita conferma a questa teoria.
Srgey Brin, fondatore e presidente di Google, ha offerto una nuova lettura a questi avvenimenti: con un colpo di scena, Brin ha annunciato un forte interessamento di Microsoft ad una forma di collaborazione con Google. Brin ha affermato di stare ancora valutando l'offerta.

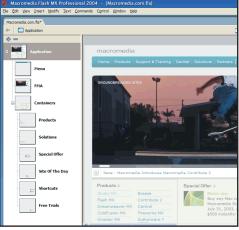
www.google.com

INTEL ENTRA NELL'ORBITA DI ECLIPSE

La grande azienda produttrice di chip entra nel consorzio fondato nel 2001 dall'IBM per creare una piattaforma di sviluppo aperta.
INTEL fornirà dei plug in che consentiranno di ottimizzare le prestazioni della applicazioni per tutta la gamma dei suoi processori. INTEL sarà rappresentata in Eclipse da Khazam, general manager della Software Products Division, che ha sottolineato l'impegno di INTEL nel supportare gli sviluppatori al fine di ottenere applicazioni più performanti. INTEL porterà su Eclipse la vasta esperienza maturata nel campo dei compilatori e nei misuratori di prestazioni.

www.eclipse.org

MACROMEDIA STUDIO MX 2004

Sono attese grandi novità per la prossima versione della suite di Macromedia. Saranno aggiornati Dreamweaver, Fireworks e Flash, ma soprattutto quest'ultimo godrà dei miglioramenti più sostanziali. Il tanto atteso supporto per i Web Services sarà finalmente nativo, così come la disponibilità a collegarsi con fonti di dati eterogenee, grazie all'integrazione del Data Connection Kit. La gestione del collegamento ai dati può essere effettuata sia attraverso ActionScript che per via visuale, mentre per il collegamento a Web Services avremo a disposizione una serie di schede che renderanno praticamente automatico l'utilizzo di servizi esterni. Anche l'occupazione di banda risulterà migliorata grazie al Data Source Shadowing che consente ottimizzare lo scambio di informazioni fra client e server, utile soprattutto nell'aggiornamento dei database. Il cambia-

I moduli sono la novità più ghiotta.

SUN E SUSE ALLEATE LINUX

Le due aziende hanno annunciato un'alleanza strategica per spingere sinergicamente Java e Linux: SuSE diventa licenziatario del source Java 2 Standard Edition e distribuirà la Virtual Machine di Java all'interno delle sue distribuzione; Sun Microsystems si occuperà di distribuire SuSE Linux Enterprise Server 8 sui sistemi Sun x86 occupandosi anche del supporto tecnico presso i clienti.
"Questo accordo attesta la fase di accelerazione impressa alla distribuzione di Java e il supporto di Sun nei confronti della comunità aperta", ha dichiarato Jonathan Schwartz, vicepresidente software di Sun Microsystems. "Stimandone le competenze, Sun accoglie SuSE nella comunità Java con l'augurio di una fattiva collaborazione per la crescita del mercato globale. Tramite questa collaborazione e il continuo impegno di Sun verso i sistemi aperti, i clienti potranno beneficiare di maggiore scelta e innovazione".

www.suse.com

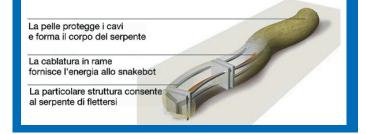
Realizzare presentazioni sarà più facile.

mento più importante risulta essere sicuramente l'introduzione dei moduli una metafora che va ad affiancare la linea temporale di Macromedia Flash. In pratica, avremo a disposizione dei form del tutto simili a quelli presenti negli ambienti RAD (Visual Basic in primis). Gli utenti che lo vorranno potranno dunque ignorare li linea temporale che tiene legato Flash alla sua origine di generatore di filmati: con l'introduzione dei moduli, Flash MX si propone come il più temibile concorrente di Microsoft per gli ambienti RAD. Lo scopo dichiarato di Macromedia è proprio quello di intercettare i milioni di sviluppatori VB 6, disorientati dalla complessità di VB.NET e ancora alla ricerca di uno strumento più aggiornato che possa eguagliare, per semplicità e flessibilità, il loro amato Visual Basic.

www.macromedia.it

STRISCIA IL ROBOT-SPIA

Un laboratorio britannico è alle prese con un nuovo genere di robot che imita il movimento dei serpenti. La sua particolarità è quella di riuscire a spostarsi anche se danneggiato. Il software riconosce infatti il tipo di danno e modifica di conseguenza i movimenti, in modo da consentire lo spostamento su terreno. Indicato soprattutto per i campi di battaglia: grazie all'assenza di ruote il suo profilo risulta particolarmente basso, riducendo la possibilità di essere individuato dal nemico. Un passo importante nella robotica dove, fino ad ora, l'integrità era condizione essenziale per un'azione efficace. Il mondo animale ci insegna ad esempio che un cane ferito ad un zampa continua a muoversi utilizzando le altre ancora integre. A fianco della innovativa sezione meccanica, è interessante notare l'algoritmo "genetico" creato ad hoc, che ha l'obiettivo di evolvere continuamene per adattarsi alle nuove situazioni.

www.bae-systems.com

CONTO ALLA ROVESCIA PER LONGHORN

Nel mese di ottobre, ad alcuni sviluppatori sarà data la possibilità di dare una prima occhiata al prossimo sistema operativo di casa Microsoft. E' stata definita una "developers preview" e, pur non essendo ancora una beta version, consentirà di sperimentare sia il look&feel, sia l'utilizzo di numerose nuove API. Tra le novità più attese del nuovo sistema operativo, ci saranno proprio le "managed API" che, pur consentendo un accesso diretto alle principali funzione del SO, impediranno il crash del sistema, attraverso una supervisione continua delle attività. Per la versione finale del sistema operativo si dovrà aspettare ancora molto: all'inizio, l'uscita sul mercato era prevista per il 2004, mentre le ultime stime danno il 2006 come data più probabile. Insomma, dobbiamo pazientare ancora parecchio.

L'interfaccia si presenta ricca di novità.

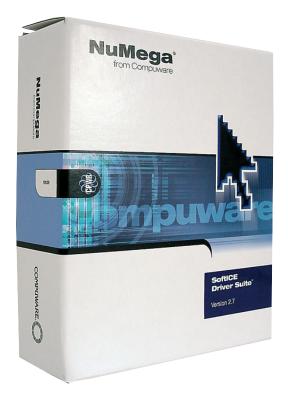
www.microsoft.com

WS-I RILASCIA IL BASIC PROFILE 1.0

La Web Services Interoperability Organization, il gruppo di aziende che ha lanciato i WS capitanato da IBM, Microsoft e BEA, ha annunciato il rilascio ufficiale delle linee guida su come utilizzare le specifiche per i Web Services al fine di sviluppare Web Services capaci di interagire con altri servizi. Anche se risulta impossibile garantire al cento per cento l'interoperabilità di un particolare servizio, il rispetto del profilo proposto garantisce la soluzione dei più comuni problemi sinora rilevati nell'esperienza comune degli sviluppatori.

www.ws-i.org

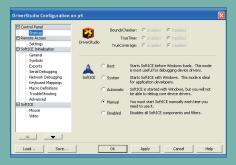
SoftICE Driver Suite

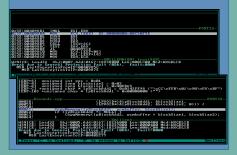
Uno strumento di vitale importanza per gli sviluppatori, che racchiude, in un unico pacchetto, una collezione di tool che semplificano la distribuzione, il debugging e il testing dei Device Driver e delle applicazioni progettate per sistemi Microsoft Windows.

Compuware Corporation ha rilasciato al pubblico, da diversi mesi, la versione 2.7 della Driver Suite, un pacchetto completo che integra una serie di strumenti in grado di cambiare la vita degli sviluppatori di device driver in ambiente Windows. La suite è d'aiuto, infatti, sia ai programmatori che agli ingegneri nel creare progetti completi e nel seguire le varie fasi di sviluppo, attraverso un ambiente di lavoro completo e ricco di funzionalità, in grado di produrre device driver che rispettano gli standard WHQL *(Windows Hardware Quality Labs)*. Si può quindi tranquillamente affermare che Driver Suite rappresenta lo "stato dell'arte" per ciò che riguarda il debugging e la programmazione a livello kernel.

LA SUITE NEI DETTAGLI

Vediamo più da vicino quali sono i componenti che formano la Driver Suite e, per gli sviluppatori che si avvicinano per la prima volta a queste tecnologie, analizziamo i dettagli e le caratteristiche di ogni singolo tool:

- **SoftICE -** SoftICE è il "Debugger" per antonomasia (con la "D" maiuscola….prego!). Si tratta infatti del pluridecorato tool che consente di scandagliare a fondo, nei meandri delle istruzioni in linguaggio macchina, ogni sorta di applicazione, libreria, driver e componente del sistema operativo (sarà per questo che è utilizzato dagli hacker di mezzo mondo?). SoftICE deve la sua grandezza al fatto che è in grado di girare a livello "kernel-mode", a differenza degli altri tool di

SoftICE all'opera

1 Al termine dell'installazione si può decidere se caricare SoftICE manualmente o se farlo caricare come servizio di sistema o addirittura se usare il loader in fase di boot. Questa scelta è legata all'uso che si vuole fare del debugger: per tracciare applicazioni in kernel-mode e a basso livello è ovviamente richiesto l'opzione Boot.

2 Il Symbol Loader di SoftICE consente di indagare a fondo, ed esplorare il codice assembly di ogni applicazione o file eseguibile, sia esso di tipo EXE, DLL o VXD).
Il tool consente, inoltre, allo sviluppatore, di potersi agganciare ad un qualsiasi processo in esecuzione.

3 Il debugger di SoftICE in azione: quando il servizio è attivo, premendo CTRL+D si può attivare in qualsiasi momento la console di debugging, che mostra lo stato dei registri, le istruzioni assembly, le chiamate alle API e lo stack dei dati. Usando il comando BPX è possibile impostare i punti d'interruzione (breakpoint).

analisi e di controllo del software che girano in "application-level", cioè come una qualsiasi altra applicazione di Windows. Questa caratteristica consente a SoftICE, ad esempio, di eseguire il debugging di driver SYS, VxD e addirittura di processi di sistema e applicazioni che girano a "ring-3". SoftICE è stato espressamente progettato per lavorare interponendosi fra il sistema operativo e l'hardware, in modo da poter osservare e tracciare le interazioni dei drivers e intervenire in caso di crash e "schermate blu" d'errore.

- **DriverWorks -** DriverWorks include un framework completo per la distribuzione di device driver sia per Windows NT, sia per sistemi Win32 generici, utilizzando il modello WDM *(windows driver model)* che è anche supportato da Windows 98/ME e 2000. DriverWorks include un sofisticato tool per la generazione rapida di codice (DriverWizard, mostrato in Fig. 1) attraverso l'uso di classi e librerie predefinite in linguaggio C++.

- **DriverNetworks** - DriverNetworks è una suite per la distribuzione di network driver per sistemi Windows. Si tratta di una serie di classi C++ utili per la creazione di NDIS driver e di client TDI, che consentono non solo di creare device driver per schede e dispositivi di rete, ma anche di realizzare progetti di protocol driver per il filtraggio in tempo reale dei pacchetti (sniffer, firewall, ecc.).

- **VTOOLSD -** Si tratta di uno strumento per la distribuzione di driver VxD in ambienti Win9X (Windows 95 e 98). Anch'esso, come gli altri tool, include una serie di classi e librerie predefinite per la generazione rapida del codice dei device driver. Sono allegati numerosi esempi e template di driver che possono essere usati come punto di partenza per la creazione di nuovi progetti.

NOVITÀ DELLA NUOVA VERSIONE

Una delle funzionalità più splendide introdotte dalla nuova Driver Suite è l'architettura host/target, piattaforma che consente allo sviluppatore di eseguire il debug e il test dei device driver presenti su un certo computer (chiamato *target*), pur trovandosi su un computer remoto (*host*), fisicamente lontano dalla macchina target. Questo consente, ad esempio, di evitare l'interferenza del debugger sullo schermo e sull'interfaccia grafica del computer target; le connessioni supportate da questa architettura sono il tradizionale TCP/IP (attraverso reti LAN e Internet) e il collegamento diretto tramite cavo seriale. Ma il punto di forza e la grande novità del pacchetto riguarda senz'altro SoftICE: con la release 2.7 finalmente viene pienamente supportato il sistema Windows XP, assicurando piena compatibilità del debugger col nuovo kernel progettato da Microsoft.
Per il resto delle novità, ecco un elenco completo e dettagliato per ciascun componente della suite:

Fig. 1: Il Wizard per la progettazione di un driver.

DriverWorks e DriverNetworks

- Aggiunto il supporto di NDIS 5.1
- Migliorato il supporto per la distribuzione di WDM driver
- Le librerie gestiscono ora l'exception handling nel kernel

SoftICE

- Visualizzazione del numero di processori logici nei processori con tecnologia Hyper-Threading
- Consente all'utente di visualizzare e salvare gli MSR *(model specific registers)* disponibili
- Aggiunto il supporto per Windows XP e il nuovo kernel

TrueTime Driver Edition

- Aggiunto il supporto per USB 1.x, NDIS 5.1 (inclusi driver Miniports) e il protocollo IEEE1394 (Firewire)
- Possibilità di misurare e monitorare le performance dei display driver e print driver.

Maggiori informazioni sono reperibili sul sito del produttore Compuware Corporation (*www.compuware.com*).

SoftICE Driver Suite

Produttore: Compuware SPA
Distributore italiano: Compuware SPA
Sito: http://www.compuware.it
Info: Tel. 800783667

Microsoft®

Tech·Ed 03

10° Anniversario

Barcellona, bella e calda come sempre. Di nuovo questa splendida città ha ospitato il TechEd: un connubio perfetto fra il più importante evento europeo di Microsoft ed una delle più affascinanti città del Mediterraneo. Sarà forse il caldo, ma questo nostro mare che ha unito culture e popoli diversi, mi ricorda il ruolo di XML nel mondo dello sviluppo: mondi separati che condividono la loro ricchezza e le loro culture, grazie ad un unico meraviglioso elemento.
Le civiltà mediterranee come antesignane dei Web Services… mi sembra davvero troppo: per la prossima settimana niente vino!

PRECONFERENCE

Il giorno di preconference lo dedico al ripasso del Framework e a gironzolare un po' per la fiera. Rispetto allo scorso anno ho l'impressione che ci siano meno espositori, o forse è solo la mancanza di folla, che puntualmente si presenterà domani, primo giorno ufficiale del TechEd. Anche quest'anno il dispiegamento di mezzi per permettere le funzioni vitali degli attendee è impressionante: nutrizione e navigazione sono garantite. Centinaia di PC ed una rete WiFi che, anche se un po' a singhiozzo, ha fornito di byte le migliaia di sviluppatori presenti.

Fig. 1: Parthasarathy assiste compiaciuto alla dimostrazione del Web Service di Vodafone che, con l'ausilio di MapPoint, permette di tracciare le coordinate di un telefonino in movimento.

KEYNOTE

In fila e di corsa per arrivare all'apertura della sessione plenaria: nessuno voleva perdersi la keynote del decennale. Solita atmosfera surriscaldata e soliti megaschermi, appena ingentiliti da un lungo e simpaticissimo spot che suggeriva la bella idea di un'azienda, la cui missione fosse di permettere ad ognuno di esprimere il proprio potenziale.
Jean Philippe Courtois, CEO di EMEA e padrone di casa, ha fatto il punto della situazione ripercorrendo brevemente le tappe che hanno portato Microsoft tra i protagonisti assoluti del mondo dello sviluppo e ha fornito il dato chiave per il .Net Momentum nella zona di sua competenza (Europa, Medio Oriente e Africa): quasi mezzo milione di sviluppatori che utilizzano strumenti Microsoft. Courtois ha sottolineato l'importanza del TechEd per Microsoft che può avere un riscontro diretto sui dubbi e sulle necessità avvertite dagli sviluppatori. L'attenzione alle domande rivolte dagli attendee agli speaker, dobbiamo dire che conferma questa impressione.

Fig. 2: Lo stand della Intel era particolarmente ricco, tra le cose più interessanti delle librerie grafiche che spiccavano performance fuori dal comune. www.intel.com

Courtois ha quasi subito lasciato campo libero a Sanjay Parthasarathy, il cui ruolo in Microsoft ricorda, per lunghezza, un titolo nobiliare spagnolo: "Corporate Vice President of the Microsoft Platform Strategy & Partner Group".
Parthasarathy entra subito nel vivo, spiegando come si pronuncia il suo nome e quale sarà il futuro della programmazione: integrazione.
Questa è la parola chiave su cui Microsoft ha puntato: sia per espandersi (obiettivo che condivide con la totalità delle aziende) sia per convivere con soluzioni diverse dalle proprie. Nel corso del TechEd si è ad esempio avuto l'impressione che Java, pur rimanendo il più temibile avversario, non è più visto da Microsoft come il diavolo, se è vero che numerose sessioni tecniche (peraltro affollatissime) sono state dedicate alla interoperabilità fra J2EE e .Net. Ovviamente, parlando di interoperabilità, non si può non parlare di Web Services che, una volta di più sono stati protagonisti di questa

keynote: numerose demo hanno illustrato come questa tecnologia si avvii ormai alla maturità e sia sempre più adottata dalle aziende per risolvere problemi reali. Una demo, in particolare, ha colpito la fantasia di tutti i presenti: un web services di Vodafone che accettava come parametro il numero telefonico di un cellulare e restituiva, in tempo reale, la posizione geografica dello stesso. L'esempio illustrato si avvaleva anche di MapPoint per la visualizzazione grafica, mentre tutto il codice era in C#. I risultati dell'interrogazione sono stati mostrati anche su un PocketPC, sottolineando come le uniche differenze di codice nella versione Pocket riguardassero la visualizzazione della mappa. Non sono stati forniti dettagli sulla disponibilità effettiva del sistema, ne' sono stati resi noti i dettagli relativi alla gestione della privacy (in Italia ad esempio solo le forze dell'ordine possono utilizzare le triangolazioni delle cellule GSM per tracciare la posizione degli utenti), se lo scopo era incuriosire e meravigliare: l'obiettivo è stato centrato!
Parthasarathy ha infine illustrato come gli attuali sforzi di Microsoft siano tesi a rendere effettivo il passaggio dagli XML Web Services ad un'architettura direttamente orientata ai servizi, architettura per cui è già pronta una nuova sigla SOA (Services Oriented Architecture).

UN PREMIO ALL'IMMAGINAZIONE

Per dimostrare che i web services sono una tecnologia pronta e matura, non c'è niente di meglio che esempi ed applicazioni concrete.
A tal fine il TechEd di Barcellona ha ospitato la finale di Imagine Cup: un nuovo concorso, indetto da Microsoft per premiare gli studenti più creativi nel trovare una soluzione che traesse vantaggio dai web services.

I team finalisti presenti a Barcellona erano 15 e si sono divisi un montepremi di 50.000$. Vincitore è stato uno sviluppatore che ha risolto un suo curioso caso personale. Figlio di un ristoratore del Nebraska, ha preso atto che nel suo locale camerieri e cuochi parlavano lingue diverse, e le ordinazioni avevano dunque una vita difficile. Per risolvere la questione, Tu Nguyen ha previsto una soluzione basata su PocketPC e web services: l'ordinazione viene presa dal cameriere attraverso un PocketPC configurato con la sua lingua. Un web service la prende in consegna e la trasmette in cucina già tradotta nella lingua propria del cuoco che dovrà occuparsi di preparare il piatto. Che dire? Geniale, ma a Napoli abbiamo già inventato il linguaggio dei gesti per capirci a dispetto di lingue diverse.

Personalmente, ho trovato molto più interessante la soluzione piazzatasi al terzo posto: quattro studenti di Singapore, attraverso un interessante mix di tecnologie, hanno realizzato un prototipo di carrello "intelligente" per supermercati. Equipaggiato con un PDA connesso wireless e di un lettore di codice a barre, il carrello offre una molteplicità di funzioni. Tanto per cominciare, il display può indicare la mappa per raggiungere un determinato prodotto poi, attraverso un touch screen, è possibile indicare la lista della spesa, lista che viene mano a mano spuntata semplicemente passando i prodotti che mettiamo nel carrello davanti al lettore di codici a barre.

La lista può anche essere inviata da casa utilizzando Internet e, mano a mano che riempiamo il carrello, il display ci aggiorna istantaneamente su quanto stiamo spendendo: grande

Fig. 5: Intel ha vinto la gara del gadget più simpatico. Un grazie di cuore a Lorenzo Di Palma per questa ed altre foto... e per le sue lezioni di pirateria.

Fig. 3: Le sessioni tecniche, cuore della manifestazione.

Fig. 4: Le sessioni che ne illustravano le potenzialità erano sempre affollate.

Fig. 6: La rete wireless era una tentazione fortissima per tutti quelli che assistevano alle sessioni tecniche.

Fig. 7: Compuware presentava DevPartner Studio, un efficace tool perfettamente integrato in Visual Studio .NET e che permette di semplificare il debug e l'ottimizzazione del codice. www.compuware.com

idea per evitare brutte sorprese alla cassa, ma clamoroso autogol per il supermarket che dovrebbe acquistare il carrello e che vedrebbe forse ridotti gli introiti. Vediamo i vantaggi che vanno a compensare questo handicap. Innanzitutto, lo smart cart può essere un eccezionale veicolo pubblicitario: informare su determinati prodotti, o su particolari promozioni, proprio mentre il consumatore è davanti allo scaffale sarebbe una grande occasione per i pubblicitari inoltre, lo staff del supermercato può essere aggiornato in tempo reale, via mms, sui prodotti che cominciano a scarseggiare sugli scaffali. Insomma, questa soluzione ci ha veramente entusiasmati e volentieri ne parleremmo più a lungo… magari nei prossimi numeri di ioProgrammo.

260 SESSIONI

Anche quest'anno le sessioni interessanti erano moltissime… e quasi sempre in parallelo! Da più parti ho sentito il desiderio di un TechEd che duri più giorni e con meno sessioni contemporanee. A parte questa critica, devo dire che la maggior parte delle sessioni presentava spunti interessanti e l'estrema disponibilità degli speaker ha permesso a molti sviluppatori di risolvere dubbi e problemi che avevano messo in valigia prima di partire.

Le sessioni più interessanti erano in genere quelle dedicate ai trucchi per utilizzare l'ambiente ed il framework e quelle dedicata alla Service Oriented Architecture. Proprio dell'attuale stadio evolutivo di SOA si è parlato con Gianpaolo Carraro (Senior Applications Architect di Microsoft): la sicurezza è in cima alle priorità di Microsoft e sono in via di definizione tutti gli standard che garantiranno l'accesso sicuro ai servizi distribuiti. Ancora molta strada c'è invece da percorrere sul piano delle transazioni e sull'affidabilità. L'evoluzione di questo sistema ha sempre come punto di riferimento un'architettura incentrata sui servizi che, dal punto di vista degli sviluppatori, significa cominciare a ragionare a partire dai contratti (e, in un certo senso, dai casi d'uso), piuttosto che dalla logica business.

Fig. 8: SAP presentava NetWeaver una piattaforma di integrazione aperta che consente di integrare le soluzioni SAP, Microsoft .NET e J2EE (con WebSphere di IBM), attraverso una intelligente declinazione del concetto di Web Service. Netweaver offre una visione centralizzata della gestione dei servizi che consente di semplificare tutto il ciclo di vita delle applicazioni Web. Di assoluto rilievo le soluzioni raggiungibili con SharePoint. www.sap.com

Fig. 9: L'incontro con Gianpaolo Carraro (Senior Applications Architect) – Il suo raggio di competenza è molto ampio e si estende particolarmente nel campo dell'interoperabilità: Service Orientated Architecture, Java 2 Platform Enterprise Edition (J2EE), .NET Interoperability, Smart Client Architecture.

Fig. 10: C'era anche un'area relax all'aperto, presa d'assalto in ogni pausa fra una sessione e l'altra.

NUOVE OCCASIONI

Tra le sessioni che più hanno stimolato la mia curiosità, devo senz'altro citare quelle riguardanti la programmazione del prossimo Office 2003. Come molti di voi già sapranno, la nuova suite per ufficio di Microsoft è completamente "XML-centrica": i documenti, che siano Word o Excel, hanno sempre dietro del codice XML. Le opportunità che si aprono sono moltissime e vanno per lo più in due direzioni: da un lato il documento stesso non è più la "tomba" dei dati che contiene ma, grazie alla natura autodescrittiva di XML, si spalanca al mondo, rendendosi disponibile a qualsiasi applicazione in grado di leggere XML; dall'altro, il documento può accogliere al suo interno riferimenti a fonti di dati esterne che lo possono rendere "vivo". Pensate ad un agente assicurativo che se ne va in giro con il suo bravo contratto da far firmare al cliente: il documento Word potrà essere continuamente aggiornato (via Web Services of corse!) con le ultime tariffe… Pensate inoltre alla mole di documenti che anche il più piccolo ufficio è in grado di produrre: con un minimo di organizzazione (e Microsoft spera con SharePoint), i dati prodotti potranno essere condivisi e riutilizzati con consistenti risparmi per l'azienda. Il salto tecnologico e "culturale" rispetto al vecchio Office è notevole, e le opportunità che si aprono sono notevoli, sia per Microsoft che per gli sviluppatori. Ormai la casa di Redmond non fa più un mistero delle grandi difficoltà incontrate nella diffusione di VB.NET. Il numero di pubblica-

Le performance di Don Box e David Platt: veri mattatori della scena!

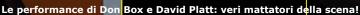



Fig. 11: L'ormai tradizionale gioco di Installshield attraeva fragorosamente i visitatori. Il prodotto di punta è AdminStudio, capace di semplificare la costruzione dei pacchetti di installazione, con una potente utility di risoluzione dei conflitti. www.installshield.com

zioni dal titolo "Da Visual Basic a VB.NET senza neanche leggermi" è incredibile, e anche ioProgrammo è caduta in tentazione. Nonostante questi sforzi e l'impegno della casa madre, la percentuale di sviluppatori che ha fatto il grande salto è decisamente modesta. La grande potenza del framework non ha evidentemente pareggiato l'inarrivabile immediatezza di Visual Basic. La disciplina e le conoscenze richieste da .Net hanno sinora costituito un ostacolo

Fig. 12: Grazie all'ottimo XMLSPY, arrivato ormai alla quinta release, Altova gode di una posizione di assoluto rilievo nel panorama degli editor XML. www.altova.com

Fig. 13: Allo stand di AMD era possibile provare i nuovi processori a 64 bit Opteron: chi voleva, aveva a disposizione una macchina con Windows Server 2003 e Visual Studio .NET. www.amd.com

che ben pochi sviluppatori hanno deciso di affrontare. Ecco dunque che uno strumento flessibile e potente come il nuovo Office 2003 può essere il giusto punto di partenza per approcciare i nuovi paradigmi della programmazione distribuita. Microsoft ci punta anche attraverso SharePoint: un sistema per creare siti aziendali intranet, sempre in modalità RAD e con la possibilità di affinare le soluzioni raggiunte con l'utilizzo di ASP.NET.

FINALE DA LEGGENDA

Il gran finale di venerdì è stato affidato a sei "leggende del software", questo il titolo che, bontà sua, Microsoft ha assegnato ai migliori specialist della sua scuderia. Si comincia con l'inarrivabile Don Box che, con il suo entusiasmo, riesce nella disperata impresa di rivitalizzare una platea annichilita da troppa tecnica e troppa Barcellona. Alle otto del mattino dell'ultimo giorno di TechEd un migliaio di attendee hanno potuto assistere ad un Don Box in forma smagliante che illustrava i vantaggi e le difficoltà di WSE 2.0 *(Web Service Enhanced)* che dovrebbe essere disponibile in beta al momento in cui leggerete queste pagine. Una slide riassumeva il suo pensiero: *WSE is for enthusiasts, The conversational platform for mission-critical apps you want to rewrite/redeploy often.* Nel complesso, la presentazione di Don Box è stata meno pregna del solito: una overview sull'attuale stato dei web services, ed una forte critica ad alcuni standard che proprio non vanno giù al vecchio Don. Ecco un'altra perla, dedicata alla difficoltà incontrate da UDDI nell'essere accettato come standard: "UDDI è la tecnologia del

Fig. 14: Rational Software e IBM avevano uno stand in comune, a suggello della loro recente unione. La piattaforma Rational assiste gli sviluppatori in tutte le fasi di sviluppo del software. www.ibm.com/rational

futuro, lo è sempre stata e sempre lo sarà!".

CONCLUSIONI

Dieci anni di crescita per il settore della programmazione. Dieci anni in cui Microsoft ha seguito, e a volte guidato, questa evoluzione e che si chiudono ora con un punto, quello di .Net. E' da qui che Microsoft ha deciso di ripartire per le prossime sfide, e non ci saranno rivoluzioni che tengano: lo sviluppo in casa Microsoft è e sarà sempre più .Net. Per restare agganciati all'evoluzione della programmazione, l'appuntamento è ad Amsterdam, 2 luglio 2004: il TechEd abbandona la città di Gaudì e ritrova la sua vocazione itinerante, chi sa che un giorno non lo si possa ospitare in Italia…

Raffaele del Monaco

Fig. 15: Davvero grande l'interesse suscitato dal Borland C# Builder, l'ambiente di sviluppo che trovate nel CD allegato a questo numero di ioProgrammo: sarà un vero antagonista per Visual Studio .NET? http://bdn.borland.com

LabVIEW 7 Express

Programma il tuo laboratorio di eletronica

Gli sviluppatori che considerano Java o Visual Basic il linguaggio di programmazione più facile e veloce da imparare e da utilizzazione probabilmente non conoscono LabVIEW (Laboratory Virtual Instrument Engineering Workbench), l'ambiente di programmazione prodotto dalla National Instrument, la cui caratteristica principale è la possibilità di realizzare applicazioni in maniera completamente visuale senza scrivere alcuna linea di codice. LabVIEW è un ambiente di programmazione potente e flessibile, orientato allo sviluppo di applicazioni per l'acquisizione e il trattamento dei segnali analogici e digitali, alla realizzazione di sistemi automatici per il collaudo, il test e il monitoraggio dei processi industriali. La semplicità d'uso e la potenza degli strumenti messi a disposizione da LabVIEW hanno portato ad una enorme diffusione di questo prodotto: il Lawrence Livermore Laboratory, il Jet Propulsion Laboratory, la NASA, il CERN di Ginevra sono esempi illustri di utilizzatori di questo moderno linguaggio di programmazione. Inoltre, LabVIEW è utilizzato sullo Space Shuttle, nei sottomarini della marina militare americana (US Navy) e sulle piattaforme petrolifere che operano nei mari del nord. Anche nelle piccole realtà

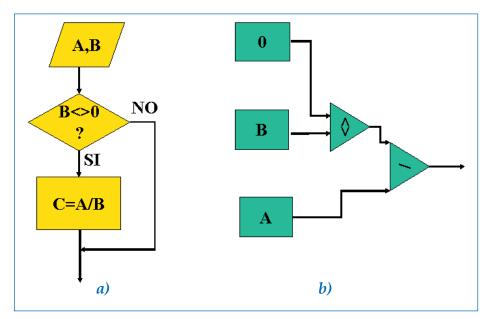
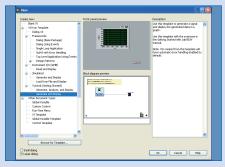
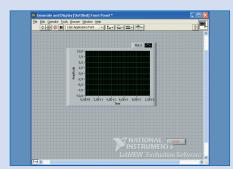



Fig. 1: **Labview offre un'alternativa al modello di programmazione di Van Newman.**

Generazione e visualizzazione di un

1 **Cliccando sul tasto *New* della finestra iniziale di *LabVIEW*, è mostratata la finestra New che consente di creare un nuovo *VI* a partire da alcuni template. Nell'elenco dei template selezionate *"VI From Template>>Tutorial (Getting Started)>>Generate and Display"*.**

2 **LabVIEW mostra una preview del front panel e del diagram block del *VI* che si sta per generare. Cliccando sul tasto *OK* avete creato il vostro primo *VI*. Provate a scorrere le voci del pannello di destra per leggere la corrispondente descrizione.**

3 **Il front panel appare con un background grigio e contiene un Waveform Graph per la visualizzazione della sinusoide e il tasto Stop per terminare l'esecuzione del programma. L'immediatezza dell'interfaccia è una caratteristica fondamentale di LabVIEW**

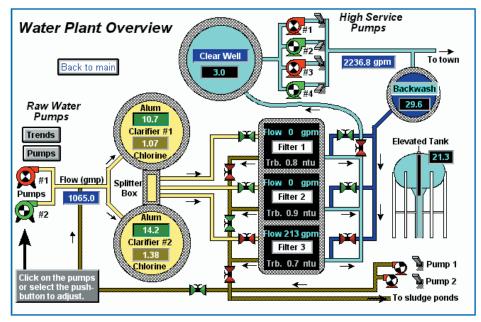



Fig. 2: Il front panel di un *VI* per il monitoraggio di un acquedotto.

industriali, molto diffuse in Italia, è molto frequente l'uso di questo ambiente di programmazione per lo sviluppo di software per il monitoraggio e il controllo come sistemi di allarme anti-incendio, sistemi di condizionamento della temperatura, ecc.

UN NUOVO LINGUAGGIO

Il cuore di LabVIEW è il linguaggio *G* (*Graphic Language*), che consente di realizzare un programma in maniera completamente visuale. Il linguaggio *G* si distacca dalla tradizionale impostazione di programma inteso come sequenza di istruzioni e sfrutta rappresentazioni grafiche degli oggetti. Due sono i principali vantaggi della programmazione visuale:

1. Le immagini aiutano la comprensione e la memorizzazione delle informazioni;
2. Le rappresentazioni grafiche hanno molto più potere rappresentativo delle parole e non hanno barriere linguistiche.

LabVIEW consente non solo di "disegnare" l'interfaccia utente, ma di "disegnare" anche la logica di programmazione senza usare costrutti cosiddetti "lineari". Mediante il linguaggio *G* la logica del programma è rappresentata da una serie di icone posizionate su di un pannello e collegate tra loro da linee di interconnessione mediante le quali "scorre" il flusso dei dati da un blocco all'altro. Il salto concettuale è abbastanza forte e, spesso, non immediato da comprendere per chi da sempre ha scritto software con i linguaggi tradizionali. I programmi possono essere realizzati in due modi: focalizzando l'attenzione sul flusso dei controlli esecutivi del programma (Fig. 1a) o focalizzando l'attenzione sul flusso dei dati (Fig. 1b). Nel primo caso il programma viene definito collegando tra loro i nodi in cui sono realizzate le varie elaborazioni. Il flusso di controllo è legato al modello di elaborazione di Von Neumann, nel quale un elaboratore è costituito da una unità di controllo, da una unità logico aritmetica, da una memoria e da unità di ingresso/uscita. Il modello a flusso di controllo è caratterizzato dalla possibilità di disporre di memoria indirizzabile (le variabili) e da un contatore che gestisce la sequenza delle istruzioni nella memoria. Nel secondo caso il modello a flusso di dati descrive il programma come una serie di nodi collegati tra loro da segmenti che specificano il flusso dei dati dai nodi che li generano a quelli che li utilizzano. In un programma a flusso dati una istruzione, come l'istruzione *dividi*, rappresentata nella Fig. 1b, sarà eseguita solo quando gli ingressi (dividendo e divisore) saranno disponibili e solo a quel punto verrà generata l'uscita. Di conseguenza non esiste nè il concetto di posizione di controllo nè il concetto di indirizzamento della memoria, in quanto un dato è presente quale risultato di una elaborazione solo finché viene utilizzato come ingresso di un'altra. Le operazioni sui dati (operazioni aritmetiche, manipolazioni di stringhe, scrittura dati su un file, tracciamento di un grafico, ecc.) sono rappresentate da icone che rendono l'idea dell'operazione compiuta. Utilizzando questo modello di programmazione è possibile concentrarsi sul flusso dei dati, mentre la sintassi più semplice rende più chiaro la logica di funzionamento del programma.

CONCETTI FONDAMENTALI

I programmi realizzati in LabVIEW sono

segnale sinusoidale

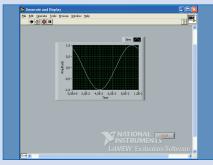
4 **Il block diagram appare con un backgound bianco e include le funzioni e le strutture di controllo del *VI*. In particolare contiene un ciclo all'interno del quale sono presenti un blocco per la generazione della sinusoide e il terminale del Waveform Graph. Il tasto *Stop* è connesso alla condizione del ciclo e ne determina la terminazione.**

5 **Cliccando sul tasto *Run* nella toolbar (il primo a sinistra), il programma va in esecuzione e mostra nella Waveform Graph un segnale sinusoidale che evolve nel tempo. Durante l'esecuzione di un programma è interessante tenere aperto il block diagram e osservare lo scorrere dei dati.**

Fig. 3: **Un esempio di diagramma a blocchi.**

chiamati *Virtual Instruments (VI)* perché imitano nell'aspetto e nelle funzionalità gli strumenti reali, ma hanno caratteristiche analoghe ai programmi realizzati in Java e C++. Un *VI* è suddiviso in due parti: il front panel e il block diagram. Il front pannel, chiamato così perchè simula il pannello frontale di uno strumento fisico, è l'interfaccia utente di un *VI*. Esso contiene i controlli che costituiscono i dati di input e gli indicatori che costituiscono i dati di output del *VI*. I controlli e gli indicatori sono tipizzati per cui è possibile utilizzare controlli o indicatori di tipo stringa, numerici, booleani, array, ecc. LabVIEW possiede una libreria molto ricca di controlli e di indicatori che consentono di costruire un'interfaccia grafica potente e user-friendly. Inoltre è possibile creare rapidamente controlli personalizzati per rendere il front panel ancora più semplice e intuitivo da usare. Il block diagram è il codice sorgente del *VI* realizzato con il linguaggio *G*. I componenti di un diagramma a blocchi sono: *VI*, funzioni predefinite, costanti e strutture di controllo di esecuzione del programma.
LabVIEW possiede funzioni predefinite molto potenti per eseguire operazioni complesse: elaborazione numerica dei segnali e delle immagini, comunicazione con dispositivi esterni, gestione di connessioni Internet, ecc. Per realizzare un programma in LabVIEW è necessario selezionare i componenti e connetterli in maniera appropriata in modo che il flusso dei dati segua il percorso desiderato. Ogni elemento del front pannel ha un terminale corrispondente sul diagramma a blocchi in modo che i dati inseriti nei controlli possano essere elaborati e i risultati mostrati sugli indicatori.
Ogni Virtual Instrument può essere usato come *subVI* (sottoprogramma) in un altro *VI*. A tale scopo ogni *VI* ha un'icona e un insieme di connettori. L'icona è la rappresentazione visuale del *VI* nel block diagram, mentre i connettori consentono di "collegare" il *subVI* con gli elementi del diagramma.
Il linguaggio *G* supporta il concetto di programmazione modulare che consente di dividere un'applicazione in una serie di sottoattività, ciascuna implementata da un *subVI* con una compito preciso

LABVIEW 7 EXPRESS

Lo scorso mese di maggio la National Instruments ha annunciato LabVIEW 7 Express. Risultato di quattro anni di intenso lavoro, LabVIEW 7 Express semplifica notevolmente la creazione di applicazioni di misura e automazione ed estende la potenza di LabVIEW anche agli FPGA embedded e ai PDA Palm OS e Microsoft Pocket PC. La novità principale della versione 7 sono gli Express VI che, progettati allo scopo di ridurre i tempi di sviluppo, tanto per gli utenti esperti quanto per i neofiti, raccolgono numerose funzionalità di misura in strumenti virtuali interattivi di facile utilizzo per le più comuni applicazioni di misura e automazione. Gli oltre quaranta *Express VI* disponibili ottimizzano lo sviluppo di task che vanno dall'acquisizione dati all'analisi di segnali fino alla gestione dei file, offrendo tutta la potenza di funzioni di misura avanzate in finestre di dialogo facili da configurare che richiedono una programmazione minima o addirittura nulla. La famiglia di prodotti LabVIEW 7 Express, disponibile per 2000/NT /XP/98, Mac OS, Linux e Sun Solaris, comprende LabVIEW Base, Full e Profsional Development System, oltre ai seguenti moduli add-on:

- Nuovo LabVIEW 7 FPGA Module - Per lo sviluppo di applicazioni per FPGA su hardware I/O riconfigurabile di National Instruments
- Nuovo LabVIEW 7 PDA Module - Per la creazione di applicazioni di misura e controllo su PDA Microsoft Pocket PC o Palm OS
- LabVIEW 7 Real-Time Module aggiornato - Per lo sviluppo di applicazioni di controllo deterministiche, real-time ed embedded
- LabVIEW 7 Datalogging and Supervisory Control Module aggiornato - Per lo sviluppo applicazioni distribuite di monitoraggio e controllo
- LabVIEW Vision Development Module - Per la creazione di applicazioni di imaging e visione artificiale.

CONCLUSIONI

LabVIEW 7 Express è un potente ambiente di programmazione per lo sviluppo di applicazioni per l'acquisizione dei segnali e il monitoraggio dei processi industriali. Consente di realizzare i programmi in maniera completamente visuale utilizzando il modello di programmazione a flusso di dati.

LabVIEW 7 Express

- **Produttore: National Instruments**
- **Sul Web: www.ni.com**
- **Nel CD: \soft\tools\Labview7eval**

INTEL® VTune™ Performance Analyzer

Spingi al massimo la velocità delle tue applicazioni!

Il VTune™ Performance Analyzer assiste gli sviluppatori nell'analisi e nell'ottimizzazione della performance del loro software. Li aiuta inoltre nell'identificare algoritmi poco efficienti e nell'avvantaggiarsi delle più avanzate caratteristiche degli ultimi processori INTEL®, inclusi Itanium® 2 , Pentium® 4 e Xeon™. La gamma di prodotti VTune™ copre diverse esigenze per diversi ambienti di sviluppo:

- VTune™ Performance Analyzer 7.0
- VTune™ Analyzer for Linux* 1.1
- VTune™ Enterprise Analyzer
- INTEL® Thread Checker 1.0

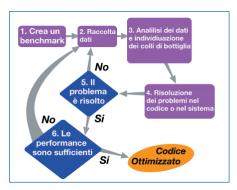



Fig. 1: Le fasi da seguire per la messa a punto del codice.

Fig. 2: I colli di bottiglia per le prestazioni sono evidenziati grazie a dettagliate informazioni su tutti gli oggetti in esecuzione.

VTUNE™ PERFORMANCE ANALYZER 7.0

E' utilizzato prevalentemente per la raccolta e l'analisi delle performance di applicazioni software. VTune™ Analyzer aiuta ad identificare gli algoritmi presenti nel codice che meglio si prestano ad essere ottimizzati al fine di aumentare le prestazioni del programma. VTune™ Analyzer è disponibile sia per i sistemi Windows che Linux. E' possibile effettuare anche analisi su sistemi per i quali non esiste una versione di VTune™, attraverso la raccolta dei dati sul sistema locale, che poi viene utilizzato per analizzare e visualizzare il risultato dell'analisi. I grafici relativi alle performance possono essere di grande ausilio per lo sviluppatore e sono visualizzati all'interno di un'efficace interfaccia utente che consente di scendere ad un fine livello di dettaglio. E' possibile identificare il consumo di risorse dei singoli thread di un'applicazione o far evidenziare le porzioni di codice che si configurano come più voraci in termini di consumo di tempo macchina. L'analisi grafica delle prestazioni è possibile solo su una macchina Windows. Per analizzare le prestazioni di un'applicazione Linux si può ricorrere all'analisi remota (usufruendo dell'interfaccia grafica della versione per Windows) oppure alla versione per Linux (senza interfaccia grafica). Vediamo brevemente le novità introdotte nella nuova versione:

- Piena integrazione con Visual Studio .NET;
- Interfaccia per la raccolta dati;
- Migliorata l'analisi per le applicazioni multithread e per i processori hyper-threaded;
- Confronto fra più processi simultanei;
- Campionamento simultaneo di più eventi.

Fig. 3: Ogni processo può essere monitorato indipendentemente.

INSTALLAZIONE

Al fine di installare correttamente il prodotto che trovate nel CD, è necessario collegarsi al link *http://www.intel.com/software/products/ distributors/creactive_eval.htm* e selezionare la voce *INTEL® VTune™ Performance Analyzer.* Lasciando il proprio nome ed il proprio indirizzo mail, in pochi minuti sarà inviato un file di licenza da copiare nella cartella del proprio PC al percorso *C:\Programmi\Common Files\Intel\Licenses.* Dal sito CreActive *www.CreActive.net/Intel* rivenditore autorizzato per l'Italia dei software INTEL®, è possibile visionare le schede tecniche con tutti i dettagli di questi tools di sviluppo e richiedere il Demo gratuito.

VTune™ Performance Analyzer

- **Rivenditore per l'Italia: CreActive**
- **Sul Web: www.CreActive.net**
- **Nel CD: \soft\tools\W_VT_P_7.0_0008.exe**

Borland C# Builder

Il massimo per sviluppare soluzioni .NET a costo zero.

Con questo ambiente di sviluppo Borland entra alla grande nell'arena degli IDE per .NET. Il framework .NET è tra le migliori piattaforme di sviluppo in circolazione ma, la scarsità di ambienti visuali gratuiti e l'elevato costo del Visual Studio di casa Microsoft ne ha finora limitato la diffusione. Con C# Builder, Borland va a colmare questa mancanza, con un IDE più semplice di VS.NET e che ne ripropone in parte la disposizione dei comandi. La gestione dei progetti si avvantaggia di alcuni comodi wizard e, grazie alla snellezza di C# Builder, anche gli sviluppatori con PC non aggiornatissimo potranno provare lo sviluppo visual su .NET. Grande attenzione è stata posta verso Web Services e Web Application, attraverso appositi strumenti che semplificano la loro creazione e la loro gestione.

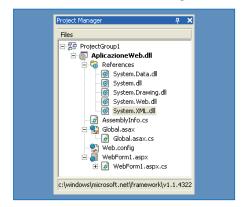



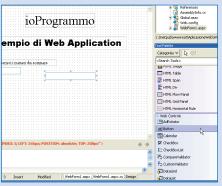
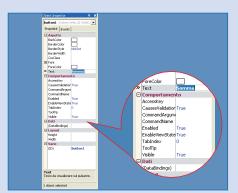
Fig. 2: **Il Project Manager consente di tenere sotto controllo tutte le risorse coinvolte nel progetto.**

Fig. 1: **All'avvio, l'ambiente ci dà il benvenuto con una serie di possibili scelte.**

L'INTERFACCIA

Pur non discostandosi molto dall'interfaccia di Visual Studio .NET, l'interfaccia di C# Builder si presente sicuramente più snella. Alcune mancanze si fanno perdonare con caratteristiche originali e utili come i tab presenti sul bordo inferiore dell'area di lavoro e che consento di switchare rapidamente fra la vista design ed il codice. Inoltre, nel caso in cui si stia svilup-

Ciclo di sviluppo di un'applicazione

1 **La creazione di una nuova applicazione Web è guidata tramite un semplice wizard. In figura vediamo il momento in cui si decide il nome ed il Web Server da utilizzare.**

2 **La costruzione dell'interfaccia Web procede attraverso delle semplici operazioni di drag&drop. Sulla destra possiamo scegliere fra una ricca selezione di componenti.**

3 **L'object inspector consente di controllare tutte le proprietà e tutti gli eventi associati ad un oggetto. In questo caso, abbiamo selezionato un pulsante del form.**

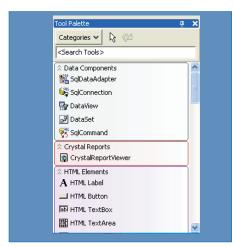



Fig. 3: **Numerosi e facilmente identificabili i componenti base offerti dall'ambiente.**

pando in ASP.NET, i tab a disposizione saranno tre, con il terzo che ci dà la possi-

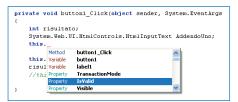



Fig. 4: **L'autocompletamento del codice è reso più intuitivo grazie all'uso dei colori.**

bilità di passare all'editor html.
In un confronto diretto con Visual Studio, bisogna comunque ammettere che C# Builder cede su molti fronti, primo fra tutti l'assenza del supporto per il Compact Framework, indispensabile per sviluppare applicazioni per PocketPC e SmartPhone.

Fig. 5: **Il collegamento a Web Service .**

DATABASE

Uno dei punti di forza dell'ambiente è quello di fornire un paradigma di sviluppo completamente design-driven che semplifica sia la fase di creazione che quella di manutenzione dell'applicazione. Fondamentale risulta essere il supporto nativo verso back-end J2EE e CORBA presente nelle versioni a pagamento di C# Builder: le aziende che vogliano integrare applicazioni già esistenti hanno finalmente uno strumento indipendente che non costringe a sposare completamente la filosofia Microsoft. Sul fronte Database, la versione Professional offre il supporto per InterBase di della stessa Borland e per MSDE 2000, mentre le versioni Architect ed Enterprise supportano anche Oracle 9i, IBM DB2 8.1.

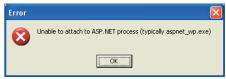



Fig. 6: **Può succedere che all'avvio di una pagina ASP.NET sia sollevato questo errore. E' sufficiente lanciare manualmente aspnet_wp.exe, così come suggerito.**

Requisiti

È necessario richiedere la chiave di attivazione collegandosi al sito della Borland.

http://www.borland.com/products/downloads/download_csharpbuilder.html#

Per installare il Borland C# Builder, i prerequisiti sono:

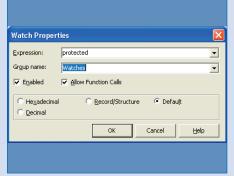
- **Microsoft .NET Framework v1.1 Redistributable**
- **Microsoft .NET SDK v1.1**
- **Microsoft Internet Explorer 6 SP1**
- **Microsoft SQL Server 2000 SP3**

Avremmo voluto includere tutti i suddetti componenti nel CD allegato alla rivista. Purtroppo un esplicito divieto di Microsoft ci ha impedito di offrire questa opportunità. Ce ne scusiamo con i lettori, ricordando che è possibile effettuare il download gratuito di tutto il software necessario presso il sito ufficiale di Microsoft.

C# Builder Personal Edition

- **Produttore: Borland**
- **Sul web: www.borland.it**
- **Prezzo: Gratuito**
- **Nel CD: \C#_Builder**

4 **L'IDE consente di operare contemporaneamente nella vista di disegno e sul codice HTML sottostante. Fondamentale durante lo sviluppo del front-end.**

5 **L'editor per il codice ricalca abbastanza fedelmente quello presente in VS.NET. Si soffre un po' la mancanza di un help esaustivo come quello di Microsoft.**

6 **Il debugger forse non è all'altezza dei progetti più completi, ma si rivela efficace per l'uso in piccole applicazioni, ambito cui la versione Personal è rivolta.**

XMLSPY Home Edition 5

La versione "casalinga" del miglior editor XML.

La fama di XMLSPY è ben meritata: tool completi ed editor velocissimi, hanno da sempre contraddistinto gli applicativi con questo nome.
Questa ennesima reincarnazione conferma le impressioni positive di tutti gli altri prodotti: semplice da utilizzare e molto flessibile, presente diverse possibilità di visualizzazione del codice.
È possibile validare documenti XML sulla base di schemi DTD/XML, effettuare delle trasformazioni XSL e personalizzare pesantemente l'interfaccia utente.
Molto valide le funzioni di autocompletamento del codice ed efficace l'help sintattico.
Eccellente il supporto ai Web Service, grazie ad un generatore di documenti WSDL.
Interessante l'interazione con Excel: attraverso dei semplici copia e incolla è possibile importare ed esportare dati XML, utilizzando la vista database di XML SPY.
Chi è alle prime armi con XML e chi deve utilizzarlo quotidianamente, troverà nella *Home Edition* di XMLSPY un valido ausilio.
Versione di prova valida trenta giorni.

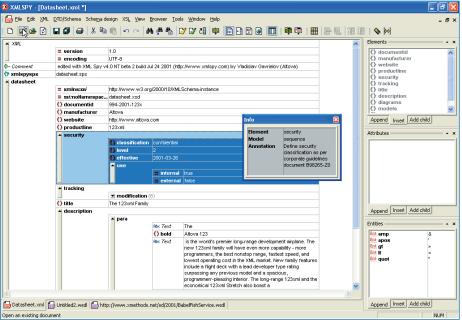
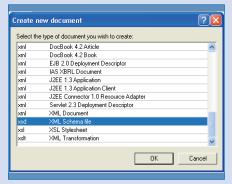
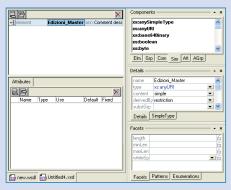



Fig. 1: La gestione dei file WSDL risulta semplificata.

XMLSPY Home Edition 5

- **Produttore: Altova**
- **Sul web: www.xmlspy.com**
- **Prezzo: $ 99.00**
- **Nel CD: \soft\tools\XMLSPYHomeComplete5.exe**

Creare un nuovo file Schema

1 Cliccare sulla voce *New* dal menu *File* e selezionare nel dialog che appare la voce *.xsd W3C XML.* Un file Schema vuoto apparirà nella finestra principale.

2 Apparirà uno schema vuoto e ci verrà chiesto di indicare il nome dell'elemento *Root*. L'elemento root avrà visibilità blobale.

3 A questo punto saremo pronti a popolare il nuovo documento. Per dare un nome allo schema, dal menu *File* scegliere *Save as.*

Java 2 SDK Standard Edition 1.4.2

Tutto quello che serve per realizzare applicazioni Java.

L'ambiente di sviluppo Sun, negli ultimi anni, si è imposta come la prima scelta per i programmatori che lavorano in ambito multipiattaforma.

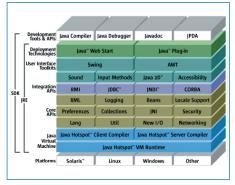



Fig. 1: Struttura della piattaforma Java.

In questa versione, nuove funzionalità e migliori prestazioni arricchiscono l'ambiente.
Rich client application e Web Services sono i campi in cui sono più evidenti i miglioramenti.
Tra i miglioramenti che più faranno gola agli sviluppatori ci sono sicuramente quelli inerenti Swing.
Davvero ghiotti i due nuovi look&feel: GTK+ e, finalmente, Windows XP. Anche le applicazioni Java possono così integrarsi pienamente nell'ambiente visuale del più recente Windows.
Anche GTK+ risulta molto interessante: attraverso un semplice resource file, è possibile settare i parametri fondamentali del look&feel.
Sempre in ambito Swing, è da notare la pesante cura dimagrante cui è stata sottoposta *JFileChooser*, che risulta essere ora molto più veloce della precedente versione, in alcuni casi anche 300 volte più veloce!

Java 2 SDK Standard Edition 1.4.2

- **Produttore: Sun Micorsystems**
- **Sul Web: www.java.sun.com**
- **Prezzo: Gratuito**
- **Nel CD: \soft\tools\j2sdk-1_4_2-windows-i586.exeXMLSPYHome Complete5.exe**

REALbasic for Windows 5.2

Crea e compila applicazioni per Windows e Mac.

Un ambiente di programmazione che rende disponibile, anche ai meno esperti, la possibilità di sviluppare applicazioni in pochissimo tempo, grazie anche alla ricca documentazione, ai numerosi tutorial e agli esempi inclusi. Oltre ad offrire la possibilità di importare codice e form da Visual Basic, REAL basic consente di compilare le applicazioni sviluppate, oltre che per Windows, anche per Mac OS 8, Mac OS 9 e Mac OS X. In questa minor release sono stati introdotti numerosi miglioramenti, il più importante dei quali riguarda l'estrema riduzione dei tempi di compilazione.
Versione dimostrativa valida trenta giorni. Al primo avvio è necessario cliccare su "Get a demo Key" per ottenere una chiave valida.

Fig. 1: Una semplice applicazione di esempio.

Fig. 2: L'editor di codice.

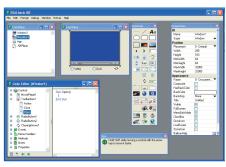



Fig. 3: L'IDE di Real-Basic.

REALbasic for Windows 5.2

- **Produttore: REAL Software, Inc.**
- **Sul Web: www.realbasic.com**
- **Prezzo: $ 99.95**
- **Nel CD: \soft\tools \REALbasicSetup.exe**

PowerVista Bridge Standard Edition 2003-1

Un mago per le Database Application

Un ambiente per la creazione di DB-Application che, grazie ad un ampio e sapiente utilizzo di Wizard, assicura una incredibile rapidità di sviluppo.
Il time-to-market risulta dunque ridottissimo e, grazie al supporto verso tutti i più diffusi DBMS (Oracle, SQL Server, Access, Informix, DB2, Interbase e altri), PowerVista potrà essere utilizzato con profitto da un ampia schiera di sviluppatori. Le applicazioni realizzate con PowerVista possono essere utilizzate indifferentemente da utenti singoli, in ambienti LAN o appoggiandosi a Internet. Versione di prova valida quattordici giorni.

Nel CD: pvbt2003.exe

OnTime Defect Tracker (Windows Edition) 2.1

Traccia, gestisce e aiuta a risolvere i bug

Basato su .NET e SQL-Server, OnTime Defect Tracker è un valido aiuto durante lo sviluppo di un progetto software: tutti i bug possono essere tracciati e gestiti attraverso complessi strumenti di ricerca ed analisi.
Le informazioni possono, ovviamente, essere condivise fra tutti i componenti al team di sviluppo, ma sulla base di apposite policy di sicurezza.
Versione limitata a tre utenti.

Nel CD: OnTimeSetup.msi

Poseidon for UML Community Edition 1.6

Un potente tool UML

Implementato completamente in Java, può girare su qualsiasi piattaforma.
I diagrammi sviluppati con Poseidon possono essere esportati in svariati formati (gif, ps, eps e svg), pieno supporto per il drag & drop, interessanti funzionalità per il reverse engineering di sorgenti Java, generazione automatica di codice Java. Compatibile con lo standard UML 1.3, Poseidon supporta tutti i diagrammi UML.
Gratuito.

Nel CD: PoseidonCE1_6_1Installer.exe

Natural Installer 1.0.1.77

Creare i setup per dispositivi Windows CE

Grazie a Natural Installer, possibile creare file di installazione per qualsiasi dispositivo compatibile Windows CE. Ecco un elenco delle funzioni salienti: supporto completo per installazioni multilingua, fine gestione delle varie fasi dell'installazione, editor di registro user-friendly, creazione automatica di shortcut per i programmi installati sul dispositivo target.

Nel CD: setup3.zip

Multi-Language Add-In for Visual Studio .NET 1.02

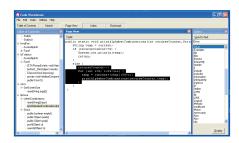
Traduci I tuoi Windows Form

Un add-in per Visual Studio .NET che consente di distribuire le nostre applicazioni in più lingue. Semplice e perfettamente integrato nell'ambiente, si occupa di ricercare tutte le stringhe testuali sia nel codice che nelle form. Le stringhe vengono poi riportate in due tabelle distinte per consentirne la traduzione e la successiva distribuzione della applicazione in formato multi-lingua. I requisiti consistono in Visual Studio .NET, Windows Installer e MDAC 2.7. Versione limitata a 50 traduzioni.

Nel CD: multilangnet.zip

Code Warehouse 1.0

Un unico grande archivio per tutto il tuo codice

Progettato allo scopo di rendere più facilmente riutilizzabile il codice che scriviamo, Code Warehouse consente di immagazzinare tutti i nostri snippet in un unico archivio. Code Warehouse si occupa di passare al setaccio le nostre classi o interi progetti, al fine di importarne automaticamente le classi nel database. Le ricerche sul codice immagazzinato possono essere effettuate secondo molteplici criteri: autore, data, nome delle classe o

procedura, per parole nel codice o anche in modo specifico all'interno dei commenti presenti nel codice.

Nel CD: codeWarehouse.zip

Setup2Go 1.9.8

Un piccolo ambiente gratuito per la creazione di Setup

Per quanto spartano, questo ambiente offre tutte l'essenziale per realizzare completi pacchetti di installazione Windows. Grazie ad un semplice e completo Wizard, anche chi non ha esperienza di programmazione può arrivare in poco tempo alla costruzione di pacchetti pronti per essere distribuiti. Gratuito.

Nel CD: setup2go1.exe

Setup Factory 6.0.1.2

Crea i file di installazione per distribuire le tue applicazioni!

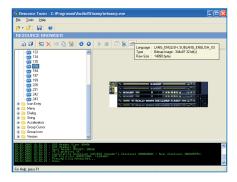
Un sistema che rende semplicissimo costruire file di installazione per le applicazioni che sviluppiamo. Sarà poi facile distribuirli via Web, e-mail, FTP, CD-ROM o Lan. Per la creazione del pacchetto di installazioni un wizard ci guida in delle semplici azioni di drag&drop e, attraverso delle intuitive strutture condizionali, sarà possibile realizzare dei pacchetti di fattura altamente professionale e capaci di adattarsi alla macchina su cui si effettua l'installazione.
Da segnalare il motore di compressione interno particolarmente efficiente e rapido.

Versione di valutazione valida trenta giorni.

Nel CD: suf60ev.exe

Resource Tuner 1.93

Esplorare e modificare le risorse contenute negli eseguibili

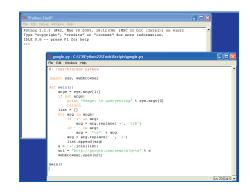
Resource Tuner è uno strumento eccellente per esplorare le risorse contenute negli eseguibili: dialog box, menu, icone, figure, toolbar, e praticamente tutto ciò che rientra nell'interfaccia di un'applicazione Windows può essere visualizzato e modificato a piacimento.

Resource Tuner gestisce un largo ventaglio di file: *EXE, DLL, SYS, MSSTYLE, CPL, OCX, SCR*, ed altri ancora. Particolarmente interessante il *Visual Style Manifest Wizard*, attraverso cui è possibile rendere le nostre applicazioni conformi allo stile di Windows XP. In questa nuova versione è possibile gestire anche documenti XML.

Trial version di trenta giorni.

Nel CD: rtsetup.exe

Python 2.2.3

Un potente tool di programmazione gratuito

Python è un eccellente linguaggio di programmazione orientato agli oggetti. La sintassi è particolarmente semplice, permette di utilizzare strutture dati di alto livello ed è possibile usufruire di un vasto campionario di librerie. Molto curato è il supporto per

XML, per i protocolli Internet e per la gestione dei dati via Web Services. Sono disponibili numerose librerie per tutti i campi dello sviluppo: dalle interfacce grafiche al digital imaging, dalla matematica all'accesso ai database e molte altre ancora. Gratuito.

Nel CD: Python-2.2.3.exe

XML Explorer 2.5

Visualizzare e modificare codice XML

Un editor che si presta a manipolare sia il codice XML che gli XML schema. L'interfaccia consente di visualizzare ed editare i documenti secondo molteplici viste: per nodi, per tabelle, come browser e come semplice testo.
Pur non brillando per velocità, l'IDE si presta ad essere utilizzato con piacere, grazie ad un'interfaccia semplice al limite dello spartano. Molto efficace la funzione di copia e incolla che permette di utilizzare la clipboard senza perdere alcuna informazione relativa alla porzione di XML copiata.
Versione di prova valida trenta giorni.

Nel CD: xesetup.exe

InstallConstruct 5.7

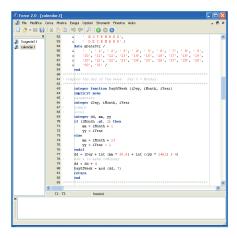
Per creare file autoinstallanti con facilità

Un'applicazione per creare distribuzioni autoinstallanti delle nostre applicazioni. Grazie ad un comodo wizard, sarà un gioco da ragazzi indicare quali file installare su qualsiasi piattaforma Windows, a partire dalla versione 3.1 (!) ad XP, passando per 95, 98, ME, NT e 2000. La distribuzione ottenuta sarà completa di programma di disinstallazione. Molto comodo il supporto per l'installazione via Web attraverso l'*Internet Component Download:* attraverso la funzione *"Search and Update"*, il pacchetto che distribuiamo può connettersi a Internet e, se presente, scaricare l'aggiornamento del prodotto.

Nel CD: ictrialsetup50.exe

Force 2.0.8

Fortran: editor e compilatore integrati

Un ambiente di sviluppo integrato che permette di utilizzare appieno il Fortran77. L'editor include tutte le più comuni caratteristiche come la colorazione sintattica, le funzioni di stampa ed altro.

Nel pacchetto è integrato il compilatore Fortran G77 ed è possibile generare applicazioni perfettamente compatibili con la piattaforma Win32.
Gratuito.

Nel CD: Force208.exe

WinDriver 6.02

Genera il codice dei driver automaticamente

Un tool che consente di accedere alle periferiche Hardware, senza la necessità di scrivere il codice dei driver. Grazie ad una serie di wizard, WinDriver riconosce l'hardware installato e genera tutto il codice C/C++ necessario a gestirlo. Le periferiche supportate sono moltissime, tra i produttori si annoverano: PLX, Altera, Cypress, QuickLogic, National Semiconductor, STMicroelectronics, Texas Instruments, Xilinx, PLDA e AMCC.
Versione di valutazione valida trenta giorni.

Nel CD: WD602.EXE

Introduzione alle tecniche che rendono sicura e privata la comunicazione.

Crittografia

Nell'ambito della sicurezza informatica rivestono un ruolo primario le tecniche crittografiche, esse permettono di rendere più protette e affidabili le nostre comunicazioni.

Accogliendo l'invito di molti lettori che trovando interessanti gli articoli scorsi sulla steganografia desideravano puntualizzare e approfondire le proprie conoscenze sulla sicurezza informatica, dedichiamo un extra all'argomento. Avevo già trattato tali tematiche nello scorso millennio, nei mesi estivi del 1999. I tempi sono quindi maturi per poter riesaminare la materia. Spero di esprimere al meglio la virtù della sintesi visto che intendo trattare la crittografia, che consta di una letteratura "sterminata", in un unico articolo. Il mio intento è mostrare ed esplorare, rispetto alla crittografia, le basi teoriche e le principali applicazioni e i software che poggiano su essa. Potremmo dire che la crittografia sta alla sicurezza informatica così come internet sta alle reti, ossia i termini *crittografia* e *sicurezza informatica* così come *internet* e *reti* sono dei sinonimi di fatto. Gli esperti sanno bene che è molto riduttivo assimilare il concetto di rete a Internet, poiché con rete si intende una vasta quantità di cose non riconducibili alla sola Internet. Analogo ragionamento vale per la crittografia che nell'immaginario romanzesco e fantasioso di chi non si occupa per mestiere di sicurezza informatica o quantomeno di sola informatica, è l'unica maniera per garantire protezione a documenti e materiale digitale. L'esistenza di altre tecniche, come l'affascinante steganografia dimostra, invece, che il settore della sicurezza è composto da tante parti di cui la crittografia è solo una di esse, peraltro la maggiore. È comunque vero che i maggiori sforzi economici e di ricerca circa la sicurezza sono rivolti verso la crittografia.

LE BASI

Abbiamo trattato la steganografia, ed in relazione ad essa, avevamo proposto una comparazione tra i due metodi formulata da Marcus Kuhn: *"La steganografia è l'arte di comunicare in modo tale da nascondere l'esistenza stessa della comunicazione. Al contrario della crittografia, in cui il nemico può rilevare, intercettare e modificare dei messaggi senza però riuscire a violare determinati livelli di sicurezza garantiti dal criptosistema …"*. Insomma, nella crittografia è ben evidente che si sta trasmettendo qualcosa di segreto, soltanto che è molto difficile "quasi impossibile" decifrarlo. Un semplice schema che esprime il processo di trasmissione mediante crittografia è proposto in Fig. 1.

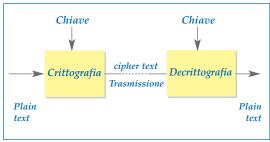



Fig. 1: **Processo di crittografia- decrittografia.**

Un testo in chiaro (*plain text)* viene cifrato in base ad un algoritmo crittografico ed in funzione di una chiave e quindi trasformato in un testo cifrato (*cipher text*). Successivamente si procede con la trasmissione, ed infine, mediante procedimento contrario, si attua la decrittografia che restituisce il testo in chiaro. Va detto che la crittografia non viene soltanto usata in funzione della sola trasmissione, si può criptare un documento per la sola necessità di renderlo protetto e riservato, senza peraltro doverlo trasmettere. Gli algoritmi di crittografia si possono classificare in due grandi famiglie: *a chiave privata* e *a chiave pubblica*. Come scopriremo, la chiave mostrata in Fig. 1 non sempre è la stessa. Nel primo caso la chiave utilizzata è la stessa sia nel processo di crittazione che di decrittazione e, tale chiave, deve essere conosciuta sia dal trasmettitore che dal ricevente, tale metodo è conosciuto anche come *metodo di crittografia simmetrica.* È ovvio che il canale di distribuzione della chiave deve essere sicuro. Questa richiesta potrebbe sembrare un paradosso: "Se conosco un canale di comunicazione sicuro allora perché crittografare?". In effetti non è così poiché un canale "sicuro" (le virgolette sono d'obbligo) si può in genere ottenere, ma con maggiore dissipazione di risorse sia in termini di tempo che di danaro. Quindi si preferisce utilizzare tali canali solo per il trasferimento delle chiavi. A proposito, le chiavi vanno cambiate periodicamente, è più affidabile! Si sceglie come canale sicuro o un corriere che trasporta materialmente da un punto all'altro la chiave o tecniche particolari che criptano la chiave secondo alcuni schemi che vengono detti a *puzzle* (come proposti da Merkle) che richiedono un gran tempo di elaborazione e

che quindi sono adatti solo per brevissimi messaggi, quale potrebbe essere una chiave. Negli algoritmi a chiave pubblica il problema delle distribuzione delle chiavi è in parte risolto. L'idea di W. Diffie e M.Hellman che svilupparono un metodo basato su un'asimmetria nella scelta delle chiavi e degli algoritmi di crittazione e decrittazione colse l'ambiente di sorpresa (il metodo è anche detto crittografia asimmetrica). In pratica, una coppia di algoritmo chiave crittografava il testo in chiaro e un'altra coppia di algoritmo e chiave (diversa dalla precedente) svolgeva la decrittazione, a questo punto si manteneva segreta solo una delle due chiavi visto che non si poteva risalire alla chiave di decrittazione a partire da quella di crittazione. Il nocciolo della questione sta nel individuare due algoritmi *C* e *D* che realizzino rispettivamente la crittazione e la decrittazione tali che *D(C(P))* = *P*, ovvero, applicando l'algoritmo di decrittazione al crittogramma (*C(P)*) si possa pervenire al *plaintext P*. Inoltre, deve essere molto difficoltoso (purtroppo non si può usare l'aggettivo impossibile) dedurre *D* da *C*. Il metodo *RSA* a chiave pubblica che tratteremo più avanti ci chiarirà le idee.

I PRIMI PASSI

Sistemi elementari crittografici, che ci introducono all'argomento, si basano su codici e cifrature. Nel primo caso si procede definendo una tabella di corrispondenza tra un codice e una messaggio segreto. Ovviamente, tale tabella deve essere riservata, cosicché inviando una sequenza di codici, si invia un messaggio cifrato. A destinazione, consultando la tabella in modo inverso, si ottiene il testo in chiaro.

Gatto	Fuggire
Topo	Distruggere il messaggio
Mela	Nascondersi
Mangia	Dare allarme
Tocca	Rimanere

Tab. 1: Ad esempio, il codice Gatto mangia Topo, equivale alla sequenza di azioni Fuggire, dare allarme, distruggere il messaggio.

Un secondo metodo, prevede la cifratura del messaggio andando ad operare sulle singole lettere di esse, ognuna delle quali viene sostituita con un'altra secondo una fissata regola, ad esempio considerando la successiva nell'alfabeto, oppure, la precedente o quella che si trova dopo 5 posizioni, insomma operando una traslazione nel sistema alfabeto (con riferimento ad esempio al codice ASCII). Si possono anche considerare permutazioni che, anziché operare sostituzioni, provvedono a "rimescolare" le lettere presenti. Si ricorda che le permutazioni di *n* elementi sono *n!* (*n fattoriale*). È esplicativa l'analisi del più antico cifrario conosciuto, quello di Cesare; questi riuscì ad ingannare i Cartaginesi cifrando testi in modo da sostituire ogni lettera con la lettera che nell'alfabeto veniva dopo tre lettere, quindi ad esempio la lettera *"a"* era sostituita con la lettera *"d"* , oppure la lettera *"q"* veniva sostituita con la lettera *"t"* e così via. Cosicché , ad esempio la parola *"attaccare"* veniva cifrata in *"dwwdffduh"* se si considera un alfabeto di cardinalità 26. Analizziamo le entità in gioco. Il testo in chiaro è *"attaccare"* il testo cifrato è l'incomprensibile (a prima vista) *"dwwdffduh"*, l'algoritmo di crittografia è la sostituzione dei singoli caratteri mentre la chiave è *"3"* che indica il numero di caratteri da traslare quando si effettua la sostituzione.

CHIAVE PRIVATA

Il più conosciuto algoritmo che implementa la crittografia a chiave privata è il DES. *Data Encryption Standard* fu sviluppato dalla IBM nel 1977 su commissione del governo degli USA e fu adottato come standard ufficiale per le informazioni non classificate. Vengono usati in modo congiunto due operazioni di cifratura: la sostituzione e la permutazione. Il testo in chiaro viene suddiviso in blocchi da 64 bit. La chiave è costituita da 64 bit di cui otto di controllo; i bit di effettiva cifratura sono 56 (alcune varianti del DES adottano chiavi di lunghezza differente). L'algoritmo consta di 19 stadi differenti. Il primo è una permutazione indipendente dalla chiave del generico blocco del plaintext, l'ultimo stadio svolge l'operazione inversa alla permutazione. Il penultimo stadio scambia i 32 bit a sinistra del blocco con i 32 bit a destra. I restanti 16 stadi operano in cascata sullo stesso blocco che, passo dopo passo, viene trasformato da una cifratura di permutazione ed una di sostituzione utilizzando sempre la stessa chiave ma sottochiavi differenti. In particolare, viene diviso il blocco da 64 bit in due parti da 32 bit ciascuna. Al generico stadio *i*, dei sedici presi in esame, le due parti dei blocchi vengono sottoposte alla cifratura di permutazione e sostituzione in modo che l'output della metà sinistra diventi l'input della destra dello stadio successivo e l'output della meta destra sia l'*or* esclusivo bit, a bit, dell'input a sinistra del passo precedente con una funzione *f* i cui parametri sono la sottochiave *Ki* e l'input a destra. La funzione *f* svolge una sequenza di quattro passi. Va ricordato che la chiave è diversa in ciascuna delle 16 iterazioni. Prima di ogni iterazione la chiave viene spezzata in due blocchi da 28 bit (infatti il totale dei bit effettivi della chiave è 56 bit), successivamente viene effettuata uno shift a sinistra, il cui offset in termini di bit dipende dal numero di iterazione, permutando tale blocco di bit si ottiene quindi la sottochiave *Ki*. Per decrittare viene usato lo stesso algoritmo in senso contrario. Esistono molti modi per potenziare il DES. Ad esempio si può camuffare il plaintext intercalando una serie di caratteri che non appartengono effettivamente al testo, si potrebbe porre, ad intervalli regolari, 9 caratteri di camuffamento e uno reale. Aggiungendo questa sorta di rumore si rende più difficile il compito all'hacker. Prima di concludere la trattazione

La sfida del n. 71

Nel numero 71 avevo lanciato una sfida che consisteva nell'individuare un messaggio nascosto nel sottotitolo dell'articolo sulla steganografia. La frase stenografata era:

"Steganografare scritture per inviare eventuali elementi trasmessivi segreti è tra le finalità maggiori, debitamente garantita d@ll'idea della immissione tra segnali statici, scelti rigorosamente. Mica futilità!";

il messaggio segreto si otteneva componendo ciclicamente le lettere di posizione 1, 2 e 3 di tutte le parole eccetto quelle più corte di 3 caratteri.
L'unico elemento di punteggiatura da considerare era il punto. Ha aiutato nella soluzione finale la presenza della @ e l'implicito intercalare uno, due e tre; come alcuni lettori mi hanno confidato. La sfida è stata raccolta e vinta da molti lettori elencati di seguito, nell'ordine di arrivo della soluzione: Andrea Battisti, Daniele, Claudio Gastaldo, Filippo Mariotti, Gianluca Cerino, Alessandro Siena, Roberto Mina, Lorenzo da Brescia, Luciano Bertoletti, Dario Candela, Nicola Pietroluongo, Daniele Albonetti, Federico De Marco, Umberto Izzo, Danilo, Paolo Terraciano, Gaetan Licci, Luca Zucconelli, Fernando Scatena, Franco Vaccaro, Joseph Scolaro, Alessio Gatti, Giorgio Politano, Emiliano Piccinini, Claudio Spiga, Antonio Pessolano, Cristian Sicilia, Stefano, Federico Finati, Michele De Meda, Bruno Ricciarello, Marco Curatolo, Andrea Alessandretti, Cristian Ghidini. Dimenticavo il messaggio segreto era:

"Scrivetemi a fabg@edmaster.it ".

Sul Web

Segnalo il seguente sito italiano:

www.enricozimmel.net

ed i seguenti siti internazionali:

www.crypto.com
www.cryptome.org

Riferimenti su ioProgrammo

I riferimenti si trovano alla sezione soluzioni. Gli articoli specifici sull'argomento sono ai numeri: 28, 29, 71, 72. Segnalo l'importante articolo di Franco Vaccaro: 30, nonché il numero: 27.

del DES va riportata una nota storica che fa capire come i governi, ed in particolare quello USA, trattino il problema della sicurezza. Quando venne sviluppato DES originariamente la chiave proposta era di 128 bit ma la *National Security Agency*, ovvero, l'ente per la sicurezza nazionale statunitense decise di ridurre i bit a 56 con un conseguente calo di prestazioni dell'algoritmo. Qualcuno potrebbe chiedersi: "una maggiore sicurezza non era più auspicabile?". In effetti, secondo alcuni osservatori esperti, il governo USA teme che lo stesso strumento sia usato per nascondere informazioni da parte di soggetti che non abbiano a che fare con lo stato. Quindi, poiché gli apparati governativi e militari sono dotati di elaboratori potenti, si preferisce poter decrittare i codici di "altri" (con attacchi che vengono chiamati di forza bruta, secondo una ricerca esaustiva della chiave) e mettersi a rischio per i propri, coscienti del fatto che sarebbero necessarie strumentazioni costose (non alla portata di tutti) per fare delle decifrazioni.

CHIAVE PUBBLICA

Il metodo di crittografia *RSA*, fu sviluppato da *R. Rivest*, *A Shamir* e *L. Adleman*. La crittografia avviene utilizzando una semplice funzione matematica (dal punto di vista algebrico, che è invece complessa dal punto di vista numerico) che, in accoppiata con una funzione simile per la decrittazione, definisce i due algoritmi distinti di crittografia e di decrittografia.
La proprietà matematica su cui si basa il tutto è la seguente:

Siano P, C, e d e n dei numeri e valga la seguente relazione:
C = Pe mod n
Che verrà adottata, per la crittazione.
Allora esisterà un numero d tale che:
P = Cd mod n

FUNZIONE DI DECRITTAZIONE

Più avanti capiremo il significato delle diverse variabili. Sfruttando la proprietà di simmetria della funzione modulo si nota che le due funzioni di crittazione e decrittazione sono l'una l'inversa dell'altra e vale la proprietà commutativa.

P = Cd mod n = C = (Pe)d mod n = (Pd)e mod n

A questo punto si supponga che *P* sia il *plaintext* o blocchi appartenenti ad esso di lunghezza massima *n*. *C* sia il *ciphertext* o blocchi di esso; *e* e *d* siano delle variabili che come vedremo dovranno essere molto grandi. L'algoritmo RSA funziona come esposto di seguito:

1. Si considerano due numeri primi qualsiasi *p* e *q* molto grandi dell'ordine di *10100*.
2. Si determina il prodotto *n=pq*.
3. Si determina il prodotto *m=(p-1)(q-1)*
4. Si calcola un numero *d* che sia minore di *m* ma che sia, rispetto a *m*, un numero primo. *d* ed *m* non hanno fattori primi in comune. *d* è dispari mentre *m* è pari.
5. Una volta scelto *d*, si determina un numero *e* che soddisfi l'equazione:

   ed = 1 (mod m)

   Ovvero *e* dovrà essere l'inverso di *d* modulo *m*.

La soluzione dell'equazione per determinare e, come detto prima, non è affatto semplice da un punto di vista numerico, soprattutto non lo è nel campo degli interi *modulo z*, in cui *z* può assumere un qualsivoglia valore intero. Esistono, comunque, dei teoremi dell'algebra che ne danno una giustificazione matematica. Quindi, per cifrare si usa la relazione *C = Pe mod n* da cui si ottiene il ciphertext *C* e per decifrare *P = Cd mod n* da cui si ottiene il plaintext *P*. Per cifrare bisogna conoscere la coppia di numeri *e*, *n* mentre per fare l'operazione inversa devono essere noti *d*, *n*. Considerato che *n* è nota, l'idea rivoluzionaria sta nel rendere pubblica, quindi nota, anche *e*, ossia la chiave per cifrare, mentre rimarrà segreta *d*. La difficoltà è insita nell'operazione di scomposizione in fattori di *n* che è un numero grandissimo. Se si riuscisse ad analizzare la fattorizzazione di *n* si potrebbero individuare i due numeri *p* e *q* e conseguentemente *m*, essendo nota *e*, anche d si potrebbe dedurre, ma come detto tale operazione è talmente complessa da poterla definire "oggi" quasi impossibile. I calcoli di Rivest dicono che la fattorizzazione di un numero con 500 cifre richiederebbe *1025 anni!!* In definitiva vi sono due chiavi, una del trasmettitore che deve essere privata e che nel caso specifico è *d*, mentre è pubblica la chiave *e* (in alcuni casi si considera pubblica la coppia *e,n*) appartenente al destinatario. A scopo puramente didattico facciamo un esempio sull'uso del metodo RSA. Per fare una prova, per semplicità di calcolo non considereremo chiavi che rispetteranno le specifiche del metodo, ovvero che siano dell'ordine di 10100, ma saranno numeri molto piccoli, per poter fare i conti con maggiore agio. Ad ogni modo vedremo come anche così si presentano "problemi". Ho fatto qualche prova che mi consentisse di individuare numeri "giusti". Ho individuato *p=5* e *q=17* che sono due numeri primi.

n=pq=85 mentre m=(p-1)(q-1)=64

Se si pone *d=13* (questa è la chiave segreta non ditela in giro) si ottiene dalla relazione *ed=1 (mod 64)* un valore di *e* pari a *5*, provare per credere! Dovremo adottare un alfabeto di cardinalità *n=45*. Con i numeri scel-

Soluzioni

ti *P* non verrà associato a blocchi di caratteri ma ad un solo carattere. Supponiamo che la *a* sia codificata con il numero *1* la *b* con il numero *2* e così via e che il messaggio da cifrare sia *"cia"* (in tema di sicurezza mi sembra una scelta azzeccata!). La crittazione si otterrà applicando $C = Pe \bmod n$. A *P* corrisponde il codice della *'c'* ovvero *3*. Applicando la formula si ottiene *C=73*. Così per le altre due lettere *'i'* e *'a'* il codice crittografato varrà rispettivamente *59* e *1*. Se si prova a decrittare si deve usare la formula $P = Cd \bmod n$. Per la lettera *'c'* il cui codice crittografato vale *73* la potenza *Cd* vale *4747758522670009868607496692953* cifra più cifra meno (scherzo il calcolo è esatto) il cui modulo di *n* è proprio *73*.

ALTRI METODI

Esistono in commercio ed in distribuzione freeware una serie, veramente cospicua, di algoritmi per la crittografia dei dati. Mi limito ad una breve presentazione dei più importanti. Tra gli algoritmi a chiave privata oltre il *DES* riveste particolare importanza *IDEA (International Data Encryption Algorithm)* poiché viene impiegato da *PGP* (metodo oggi molto usato). Fu progettato nel 1991 da *X. Lai* e *J. L. Massey*, inizialmente aveva il nome di *IPES (Improved Proposed Encryption Standard)*. Il funzionamento per linee generali, è simile al *DES* anche se va detto che la progettazione di tale metodo, a contrario di *DES*, (che fu ideato in un primo momento come dispositivo hardware) fu da subito di tipo software. La chiave è di lunghezza di 128 bit, il testo in chiaro viene suddiviso in blocchi da 64 bit che a loro volta vengono spezzettati in mini blocchi da 16 bit. Ogni mini blocco subisce otto round su cui vengono applicate 52 diverse sottochiavi da 16 bit, ottenute dalla chiave primaria da 128 bit. I round si realizzano con operazioni come *XOR*, *addizione* e *moltiplicazioni*. La decrittazione è l'operazione inversa che permette, a partire dalla chiave di crittazione, di ottenere il testo in chiaro dal *cipher text*. Le prestazioni di *IDEA* sono comparabili con *TDES* (evoluzione di *DES*). Altri metodi sono *RC2* ed il suo successore *RC4* entrambi ideati da Ron Rivest, per questi algoritmi la lunghezza delle chiavi è variabile. Maggiore sicurezza si ha naturalmente per chiavi di lunghezza maggiore a scapito della velocità di elaborazione; come *TDES* che realizza una tripla crittazione. Atri metodi degni di nota sono: *Blowfish* realizzato da *B. Schneier* che genera chiavi di lunghezza variabile fino a 448 bit a partire da blocchi di 64 bit (tale metodo è considerato tra i più sicuri in circolazione); e *Skipjack* che, secondo alcune voci, dovrebbe essere il sostituto di *DES* per le applicazioni del governo degli Stati Uniti. Questo ultimo, come i metodi citati precedentemente, crittografa blocchi di testo di 64 bit con chiavi da 80 bit. Non si hanno altre notizie su di esso dato che è sotto l'esame del governo americano e della NSA. *Safer* è stato prodotto da *Massey* la chiave è a 128 bit. Tra gli algoritmi a chiave pubblica di maggiore rilievo, oltre il citato RSA, vi sono: *DSS* che però fornisce la sola funzione di firma, il suo uso è un po' macchinoso, riveste comunque una certa importanza dato che è omologato dal governo USA; *LUC* è un algoritmo freeware.

FUNZIONI ONE WAY HASH E FIRMA DIGITALE

Una funzione *One way hash* permette di estrarre una sorta di impronta da un testo. Attraverso una funzione hash viene associato ad un testo di lunghezza qualsiasi una stringa. Ciò ci consente di avere una autenticazione del documento. In altri termini il testo, prima di essere trasmesso dal mittente al ricevente, viene sottoposto a una funzione One way hash, si ottiene quindi una stringa di lunghezza relativamente piccola che viene associata al testo da trasmettere (ad esempio di 128 byte). Alla ricezione del messaggio, si sottopone il testo pervenuto alla stessa funzione, se la stringa ottenuta è la stessa di quella di partenza vuol dire che tutto è andato per il meglio, altrimenti il testo è stato manipolato oppure, nella trasmissione, per problemi tecnici, ha subito variazioni. Quindi oltre che l'autenticità viene controllata anche l'integrità del testo trasmesso. Va detto che come tutte le funzioni hash, non è reversibile, ovvero non è possibile pervenire al testo in chiaro dalla stringa ottenuta come output della funzione (da qui *"one way"*). Inoltre, testi diversi possono produrre la stessa stringa, ciò non toglie che le manipolazioni siano pressoché impossibili. La firma digitale è un procedimento algoritmico che consente di legare un documento digitale al suo legittimo proprietario, attua la one way hash. È una della maggiori applicazioni della crittografia a chiave pubblica. Così, utilizzando tale tecnica, ad ogni documento sarà associata una firma (una sequenza di byte) che potrà anche essere apposta in coda al documento. Viene così garantita, oltre che l'autenticità, anche l'integrità del documento originale. Interessanti sono i risvolti legislativi. Per le leggi italiane le firme digitali vanno convalidate da un organo apposito, *Certification authority* che rilascia un certificato digitale. Ovviamente, rigorosi sono i controlli per garantire la corrispondenza biunivoca tra la persona fisica e la chiave pubblica.

CONCLUSIONI

Spero di essere stato sintetico e nello stesso tempo chiaro nonostante i concetti fondamentali da affrontare fossero numerosi. Ad ogni modo, al termine di questa "filiera" di tre articoli, penso che tutti abbiamo raggiunto una visione più chiara ed approfondita circa la sicurezza informatica.

Le possibilità per approfondire, tanto su web, quanto su tradizionali libri sono molteplici, per cui non mi resta che salutarvi e attendere il prossimo appuntamento.

Fabio Grimaldi

Bibliografia

- **SEGRETI SPIE CODICI CIFRATI** *C. Giustozzi, A.Monti, E.Zimuel* (Apogeo)
- **AN INTRODUCTION TO CRYPTOGRAPHY** *Molin R.A* (CRC Press)
- **CODES, CIPHERS AND SECRET WRITING** *Gardner M* (Dover pubns)
- **CRITTOGRAFIA, PRINCIPI, ALGORITMI, APPLICAZIONI** *P. Ferragina, F. Luccio* (Bollati Boringhieri)
- **MATHEMATICAL MYSTERIES** *Clawson, C. Calvin* (Plenum Press) 1996
- **MANUAL OF CRYPTOGRAPHY** *British war office* (Aegean Park Pr)
- **CAPTIVATING CRYPTOGRAMS** *J. Van Dyke* (Sterling Publications)
- **CRYPTANALYSIS A STUDY OF CHIPERS AND THEIR SOLUTIONS** *Fouche Gaines H.* (Dover Pubns)
- **CRIPTOGRAPHY AND CODING** *M. Walzer* (Sprinter Verlag)

```
SELECT
 ProductID,
 ProductName,
 CategoryName
FROM PRODUCTS INNER JOIN
CATEGORIES
ON PRODUCTS.
CATEGORYID=CATEGORIES.
CATEGORYID FOR XML AUTO

Resultset...
<PRODUCTS ProductID="1"
ProductName="Chai">
 <CATEGORIES
CategoryName="Beverages"/>
</PRODUCTS>
<PRODUCTS ProductID="2"
ProductName="Chang">
 <CATEGORIES
CategoryName="Beverages"/>
</PRODUCTS>
```

Reverse Engineering: in pratica

Costruire un crack per le applicazioni Windows

Sicurezza

In questo articolo ci occuperemo del reverse engineering applicato ai programmi e scopriremo quali sono le tecniche più utilizzate per violare le protezioni.

Il termine "reverse engineering" individua oggi tutta quella serie di discipline che hanno il fine comune di studiare un prodotto funzionante (sia esso un microchip, un software, una serie di dati o qualsiasi altra cosa) per capire come è fatto, quale è stato il suo iter produttivo e per identificare la sorgente che l'ha generato.

E' GIUSTO CHIEDERSI SE È GIUSTO?

Tutti questi scenari, molto diversi tra loro, rappresentano oggi le applicazioni consuete delle tecniche di reverse engineering e sono tutti accomunati da un modello di studio identico: si parte da uno schema blackbox del sistema da analizzare, considerato come una scatola chiusa e totalmente sconosciuta, da cui si cerca, con ogni mezzo e con diversi espedienti, di estrarre e conoscere quante più informazioni possibili su di esso, fino a capire i meccanismi e le leggi che lo governano, cosa che permette di predire il comportamento del sistema o addirittura di indirizzarlo a proprio piacimento. Rileggendo queste ultime righe sembra quasi di citare i trattati sulla sperimentazione scientifica di Galileo Galilei (redatti circa quattro secoli fa): l'osservazione del fenomeno, l'ipotesi, la descrizione matematica e l'esperimento finale. Non sono forse queste le stesse fasi di un processo di retro-ingegneria? Tuttavia, come spesso accade a tutti coloro che studiano fenomeni ignoti e oscuri a molti, la guerra operata dagli "uomini in nero", continua ancora oggi a mietere vittime: così, come nel 1600 toccava a Galileo (incriminato dalla Chiesa per i suoi studi), ai giorni nostri tocca invece ad un ragazzino quindicenne subire processi e accuse infamanti per aver "osato" analizzare e pubblicare le scoperte relative al sistema di codifica dei DVD, ritenuto segreto e inviolabile (ricordate il caso del DeCSS relativo a Joan Johaensen, nel 1999?). Siamo infatti giunti al paradosso della stupidità, con le ultime leggi approvate in materia di copyright (vedi DMCA americano e il recente EUCD) : la colpa delle insicurezze di un sistema è di chi le scopre attraverso il reverse engineering e non di chi ha progettato male il sistema…. perché le insicurezze, quando non sono visibili a tutti, è come se non esistessero! A questo punto tanto vale dire che se Newton avesse brevettato la sua famosa legge di gravitazione, oggi saremmo tutti costretti a pagare le royalty… ma la legge di gravitazione esisteva ancor prima che venisse scoperta ed enunciata al pubblico, Newton ha soltanto notato che se una mela cade, non lo fa a caso, ma secondo una legge fisica. E' forse questo un reato? Terminiamo qui le disquisizioni di carattere etico sul problema, con le quali potremmo riempire tutte le pagine di questa rivista, e spostiamo invece la nostra attenzione su un'altra domanda che molti si pongono: "perché? quale stimolo spinge una persona a smontare una Xbox fino all'ultimo chip rischiando di romperla ed un'altra a girovagare nei meandri delle istruzioni assembly di un programma?". Le risposte alla domanda sul "perché del reverse engineering" sono molteplici, ma la più veritiera è soltanto una: "per dimostrare che si può".

Quando il reverse engineering è utile...

L'avanzata del worm *MSblast/LovSan* è stata in parte contenuta grazie alle tempestive analisi realizzate da molti appassionati di reverse engineering http://www.backtrace.de/msblast_analysis.txt **Paradossalmente l'autore del worm potrebbe impugnare il DMCA nei confronti delle società antivirus, chiedendo i diritti d'autore sulla sua "creatura".**

I FERRI DEL MESTIERE

L'articolo di oggi tratta nello specifico il reverse engineering operato sui programmi, in particolar modo in ambiente Windows. La parte tecnica di questo articolo sarà divisa in tre diverse sezioni: nella prima presenteremo i tool e i programmi utilizzati comunemente dai cracker per violare le protezioni e illustreremo come ci si difende da questi; nella seconda saranno invece presentate alcune nozioni di base sull'uso del debugger e infine nella terza parte si passerà alla pratica, cercando di mettere a frutto le conoscenze acquisite. Sia chiaro non violeremo certo la protezione di alcun software commerciale, ma ci limiteremo a scrivere alcuni listati di esempio (CRACKME) sui quali esercitarci nelle operazioni di reverse engineering. Come ogni "professionista" che si rispetti, anche il reverse engineer ha le sue armi, i suoi trucchi e i "ferri" del mestiere. Come un chirurgo non può operare senza bisturi, guanti e mascherina, allo stesso modo un reverse engineer ha bisogno di debugger, hex editor e disassemblatori per compiere il suo lavoro. Le utility ed i programmi di reverse engineering sono abbastanza sofisticati, non sempre reperibili e spesso complicati da usare, ognuno

con compiti ben precisi. Il punto di partenza di un reverse engineer, dove si possono reperire informazioni e utility di ogni sorta, è senz'altro il sito Programmer's Tools gestito da Kaparo (*http://protools.cjb.net*). Per comodità conviene catalogare i programmi usati nel reverse engineering del software in sei grandi categorie, di seguito riportate con una breve descrizione dei compiti e delle funzionalità di ogni tipologia:

1) Analizzatore o Scanner

Conoscere il nemico è il primo passo di qualsiasi forma di attacco. Recuperare il maggior numero di informazioni possibili su un programma è importante per capire in che contesto si sta lavorando e se si ha a che fare con un programma qualsiasi o con un nemico ben corazzato e protetto. Se un programma è scritto in VB, non ci sogneremo mai di andare ad eseguirne direttamente il debugging col rischio di finire nella giungla di istruzioni della MSVBVM60.DLL; differentemente, per un programma scritto in C++, si potranno facilmente piazzare breakpoint sulle API di sistema più comuni. Per difendersi dagli scanner, molte casi produttrici adottano spesso contromisure protettive che hanno lo scopo di bloccare (o semplicemente di rendere complicate) le procedure di inversione. Alcune di queste contromisure sono rappresentate dai programmi chiamati "packers", come ad esempio Armadillo (www.siliconrealms.com), ASProtect (www.aspack.com) o UPX (upx.sourceforge.net) capaci di criptare e comprimere un file, lasciandolo sempre eseguibile, ma rendendolo di fatto inaccessibile a disassemblatori e debugger. Un buon scanner è in grado di riconoscere il formato di un eseguibile, indicando da quale compilatore è stato prodotto (Visual c++, MASM, Visual Basic, ecc.), quali funzioni esporta ed importa e se è stato protetto da qualche packer. Gli scanner più famosi in circolazione sono senz'altro i seguenti:

- **File insPEctor XL** *http://www.fileinspector.cjb.net*
- **Stud_PE** *http://christig.virtualave.net/ITimer/studpe.html*
- **GetTyp** *http://www.unet.univie.ac.at/~a9606653/gettyp/ download.htm*

2) Dumper e Unpacker

Quando un file viene protetto da un packer generalmente la copia originale è memorizzata in forma criptata; al momento dell'esecuzione un modulo detto loader decripta il file originale e lo carica in memoria per mandarlo in run. Questa protezione impedisce quindi ogni forma di analisi sul file EXE statico, perché il file originale non è più interpretabile da debugger e disassemblatori, in quanto nascosto. Il lavoro di un reverse engineer potrebbe quindi finire qui… tuttavia per bypassare un packer si può ricorrere ad un dumper, cioè un programma capace di creare un file fisico a partire da un processo attivo in memoria (operazione chiamata appunto dump). Il file estrapolato dal dumper rappresenta l'eseguibile puro, senza alcuna protezione,

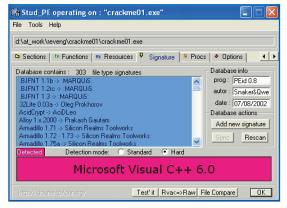



Fig. 1: **Stud_PE è forse l'analizzatore di file eseguibili più usato dai reverse engineer. Individua con esattezza il compilatore usato per creare un file EXE e riconosce centinaia di protezioni e packer commerciali.**

tuttavia spesso il dump può generare un file incompleto o allineato male (a causa delle precauzioni prese dal packer), che quindi non funziona direttamente. In questi casi bisogna ricordare che per ogni packer conosciuto è in genere reperibile in rete l'apposito unpacker o esiste una tecnica di unpacking manuale, capace di riportare alla luce l'eseguibile originale. Gli unpacker più usati sono ProcDump e GUW32, che sono di tipo generico e gestiscono diversi tipi di packer; possono essere reperiti, assieme a molti altri unpacker specifici, agli indirizzi internet *http://protools.cjb.net* e *http://unpack.cjb.net*.

3) Disassemblatore e Decompilatore

Spesso, quando è necessario avere una visione d'insieme del software in fase di analisi, si ricorre ad un disassemblatore. Si tratta di un tool in grado di ricostruire l'equivalente di un programma - in linguaggio assembly - partendo dall'eseguibile. Questa procedura produce un listato di istruzioni assembly contenente il flusso del programma e i riferimenti alle chiamate di subroutine, alle DLL importate, alle stringhe di testo e ai salti condizionati: quando si studia un particolare punto di un programma, conviene analizzare in maniera "statica" alcune routine, prima di passare all'uso del debugger (specie quando si vuole realizzare un keygen…). Naturalmente per sfruttare a pieno le potenzialità di un disassemblatore occorre conoscere il linguaggio assembly e il significato delle istruzioni a basso livello dei processori. Il Decompilatore è uno stretto parente del disassemblatore, ma fa qualcosa in più: è infatti in grado di ricostruire il sorgente completo, in linguaggio ad alto livello, a partire dal file compilato. Questa operazione è possibile solo per alcuni linguaggi, come Java o il recente formato CLR (common language runtime) introdotto da Microsoft con .NET, che producono un file compilato in pseudo-codice che viene eseguito ed interpretato da una macchina virtuale (VM). L'estrema portabilità di questi linguaggi (i cui eseguibili possono girare su diverse piattaforme) ha un prezzo che si paga con la facilità di decompilazione. Nel caso del C++ il discorso è più complicato:

Da dove partire

Il punto di partenza di ogni cracker e reverse engineer che si rispetti è senza dubbio il sito Programmer's Tools http://protools.cjb.net **dove è possibile reperire debugger, decompilatori, unpacker, utility e tante informazioni sull'arte del reverse engineering.**

Sicurezza

Costruire *un crack per le applicazioni Windows*

Fig. 2: BDASM in azione: nella foto viene mostrato il codice disassemblato del file CRACKME01.EXE, usato come "cavia" in questo articolo. Usando un disassemblatore si può analizzare nel suo insieme il codice assembly di un programma e si possono addirittura scoprire riferimenti a stringhe nascoste come in questo caso.

non è infatti possibile decompilare per intero (come avviene in Java) un eseguibile generato da un compilatore C, tuttavia esistono alcune utility in grado di ricostruire parzialmente la struttura di un programma, producendo un listato ibrido in C++/Assembly.

- **W32DASM** – Disassemblatore (*www.softnews.ro/public/cat/5/1/5-1-7.shtml*)
- **BDASM** – Disassemblatore (*http://www.bdasm.com*)
- **IDA Professional** – Disassemblatore (*www.datarescue.com/idabase/ida.htm*)
- **REC** – Decompilatore C++ (*www.backerstreet.com/rec/recita.htm*)
- **JAD** – Decompilatore Java (*kpdus.tripod.com/jad.html*)
- **DeCafe Pro** – Decompilatore Java (*decafe.hypermart.net*)
- **Reflector** – Decompilatore .NET (*www.aisto.com/roeder/dotnet*)

DeCSS, un caso controverso...

Per dimostrare i paradossi sul diritto d'autore e sul reverse engineering alcuni dimostranti hanno deciso di pubblicare il codice "incriminato" del DeCSS (quello che viola la protezione dei DVD) in ogni forma

http://www-2.cs.cmu.edu/~dst/DeCSS/Gallery.

Si è addirittura pensato di stampare delle t-shirt con l'incisione del codice del DeCSS...saranno illegali anche queste?

4) Monitor

La maggior parte di software circolante in versione trial contiene spesso una procedura a scadenza basata sulla data e sul conteggio dei giorni. Per tenere conto dei giorni è indispensabile sapere con esattezza il giorno in cui avviene la prima installazione, è inoltre necessario marcare, con qualche segno di riconoscimento, l'avvenuta installazione di un programma su un computer per evitare successive re-installazioni. Come fare per accorgersi di questi segni invisibili operati dai programmi su un sistema? In genere si tratta di operazioni di I/O che scrivono file sul disco fisso o che memorizzano chiavi nel registro di sistema.
Il modo migliore è quello di usare un monitor, ossia un programma in grado di intercettare in tempo reale tutte le chiamate in ingresso e in uscita effettuate su disco, sul registro di sistema, sulle porte (seriale, parallela) e tramite TCP/IP. I miglior monitor in circolazione sono senza dubbio quelli prodotti dalla SysInternals (*www.sysinternals .com*), freeware, che si dividono in:

- **FileMon** – Monitor per attività su file e su disco
- **RegMon** – Monitor per attività sul registro di Windows
- **PortMon** – Monitor per attività sulla porta seriale e parallela
- **TDImon** – Monitor per attività TCP/IP

5) Debugger

Il debugger è lo strumento più complicato da usare per i principianti, perché a differenza degli altri tool richiede l'interazione continua dell'utente.

Fig. 3: Una screenshot di SoftICE: il debugger può essere attivato dall'utente in qualsiasi momento premendo la combinazione *CTRL+D*. SoftICE viene caricato come servizio del sistema operativo e può essere avviato automaticamente o manualmente usando il comando "net start ntice".

Un debugger consente in sostanza di eseguire, passo dopo passo, un file eseguibile, mostrando a video le diverse istruzioni e le chiamate alle API di sistema e consentendo al reverse engineer di studiare un algoritmo istruzione per istruzione. Il debugger mostra inoltre, durante l'esecuzione, lo stato dei diversi registri della CPU, i dati contenuti nella memoria ed è in grado di intercettare momenti precisi dell'esecuzione grazie ai Breakpoint (punti di interruzione), che permettono – ad esempio – di bloccare il flusso di un programma nel momento in cui questo mostra il messaggio "Licenza non valida" o quando prova a leggere la data del sistema. I migliori debugger in circolazione sono i seguenti, tra cui segnaliamo OllyDbg che è totalmente freeware:

- **SoftICE** – (*www.compuware.com*)
- **OllyDbg** – (*home.t-online.de/home/Ollydbg/download.htm*)

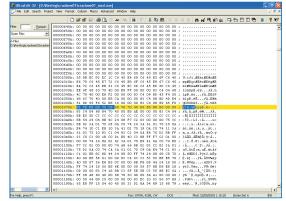



Fig. 4: L'editor esadecimale è indispensabile per portare a termine il crack di un programma. Cambiando un singolo byte nel programma (spesso quello di una istruzione *JUMP*), come nell'esempio, si può far attivare una licenza anche quando questa non è valida.

Sicurezza

Costruire
un crack per le applicazioni Windows

6) Hex Editor e Patcher
Al termine di una procedura di reverse engineering, di solito vengono apportate alcune modifiche al programma originale, cambiando alcuni bytes al suo interno. I famigerati "crack" che tutti cercano su Internet e che spesso solo i nostri amici smanettoni sembrano trovare, non sono altro che dei patcher, ovvero dei programmi che alterano uno o più byte all'interno di un eseguibile per forzare la routine di protezione. Un singolo byte, spostato ad esempio da 74 a 75 (una JE che diventa JNE), può cambiare il significato di un programma. L'hex editor è invece un tool che consente di visualizzare un file in formato esadecimale, byte per byte, e che consente di eseguire ricerche di pattern e modifiche dei codici hex. Tra i più famosi citiamo Ultraedit (*www.ultraedit.com*) e BIEW (*sourceforge.net/projects/biew*).

NOZIONI SUI DEBUGGER

Come è fatto un debugger? In ambiente Windows esistono diversi tool di debugging: SoftICE è il "Debugger" (con la "D" maiuscola, prego!) prodotto da Compuware e distribuito ora assieme alla suite DriverStudio, ma visto il costo proibitivo di questo prodotto, spesso alcuni cracker utilizzano TRW2000 (che non funziona però sotto 2000/XP) oppure il praticissimo Ollydbg, che non necessita di installazione. La differenza tra questi debugger e SoftICE sta nel fatto che quest'ultimo lavora in kernel-mode e quindi può intercettare qualsiasi chiamata del sistema operativo, anche quelle a basso livello, relative a driver e processi. SoftICE si avvia come un servizio del sistema operativo Windows 2000/XP (tramite il comando net start ntice). Quando Softice è attivo in memoria, è possibile richiamarlo in qualsiasi momento premendo CTRL+D: la prima volta che si entra nel debugger si rimane sconcertati per la schermata (Fig. 3) poco comprensibile che appare, simile nella forma, a questa:

```
EAX EBX ECX EDX ESI
EDI EBP ESP EIP [FLAG REG]
0157:406030 20 20 20 20 20 20 20 20 20 20 20 20 20
                                          20 20 20
0157:406040 20 20 20 20 20 20 20 20 20 20 20 20
                                       20 20 20 20
:0040108B 83C404        add   esp,00000004
:0040108E 5F            pop   edi
:00401096 E855030000     call   004013F0
```

Le sigle in alto (EAX, EBX, ECX, ecc.) rappresentano i registri della CPU, cioè delle celle (a 32-bit) in grado di contenere dei valori rappresentativi dello stato del processore. L'architettura di un calcolatore rispetta il modello di automa a stati di Von Neumann, di conseguenza l'esecuzione di un programma è rappresentata dal continuo modificarsi dello stato dei registri, che determina il flusso di esecuzione. Ogni registro ha uno scopo ben preciso: ECX è usato spesso come contatore (implementa spesso i cicli di for/while), EBX è usato per puntare bytes presenti nella memoria, EDX come registro dati, EAX come accumulatore e così via. Un registro molto importante è EIP (instruction pointer), che contiene l'indirizzo dell'istruzione corrente da eseguire: quando si esegue un salto ad una altra istruzione (o una chiamata), il registro EIP viene modificato per puntare alla nuova istruzione. Al di sotto dei registri troviamo quella che è chiamata "Data Window", che rappresenta una finestra – sempre aperta – su una pagina della memoria. Ogni riga è formata dall'indirizzo (segmento:offset) seguito dai 16 bytes presenti a quella locazione di memoria e dal loro equivalente in ASCII; per attivare la Data Window basta digitare il comando "DATA". Per scorrere la data windows basta usare ALT in combinazione con le frecce cursore oppure usare il comando "D indirizzo" (=dump). Nella parte centrale della finestra c'è infine il codice assembly dove ogni riga è formata da un offset, dai bytes corrispondenti all'istruzione (*opcodes*) e dall'istruzione simbolica associata. Per scorrere quest'ultima finestra si usa CTRL con le frecce cursore oppure il comando "U indirizzo" (=unassemble). Per addentrarsi nel mondo del reverse engineering è indispensabile conoscere un minimo di linguaggio assembly, il funzionamento dello stack e alcune delle istruzioni basilari (come *MOV, ADD, SUB, LEA* e i diversi salti *JMP, JE, JNE*, ecc.).

LA PRIMA SFIDA!

Per iniziare a mettere le mani su qualcosa di concreto, creiamo un primo semplice programma, che chiameremo *CRACKME01* e che cercheremo di forzare. Il sorgente della nostra prima sfida sarà volutamente scritto in forma molto semplice, proprio per aver modo di capire le tecniche di analisi. *CRACKME01* è un piccolo programma scritto in C++, che richiede in input una licenza valida (la stringa "ioprogrammo"), necessaria per proseguire l'esecuzione.

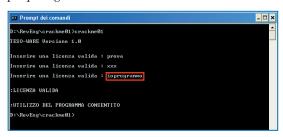



Fig. 5: CRACKME01 è un semplice programma che da console legge una stringa di testo necessaria per proseguire l'esecuzione. Il programma si avvia solo se la stringa immessa è "ioprogrammo". E' la cavia usata per provare il reverse engineering in questo articolo.

Il listato da compilare è il seguente:

```
#include <stdio.h>
#include <string.h>
void main(void) {
//password/serial number di accesso
```

Aspetti legali del reverse engineering

Come l'America anche l'Europa si sta muovendo verso l'approvazione di nuove norme restrittive che mettono al bando ogni forma di reverse engineering e che cercano di tutelare il diritto d'autore in maniera esasperata. All'indirizzo

http://www.softwarelibero.it/progetti/eucd/analisi.html

si può trovare un'analisi esaustiva dell'EUCD, la direttiva sul copyright europea che si ispira al DMCA Americano.

Sicurezza

Costruire
un crack per le applicazioni Windows

```
char* pwd="ioprogrammo";
//stringa dove memorizzare la password letta da input
char str[11];
//variabile di controllo
int check;
printf("\nTESO-WARE Versione 1.0\n\n");
do {
  printf("\nInserire una licenza valida : ");
  scanf("%s",str);
  check=strcmp(str,pwd); //confronto, licenza valida? }
while (check!=0);
//zona di accesso al programma principale
printf("\n\n:LICENZA VALIDA\n");
printf("\n\n:UTILIZZO DEL PROGRAMMA
CONSENTITO\n");
//...
//...parte restante del programma
//...
}
```

Per compilare *CRACKME01.CPP* abbiamo usato Visual C++ 6.0, generando il file eseguibile *CRACKME01.EXE* (di soli 32 KB) che sarà l'oggetto dei nostri studi e che verrà forzato. Dimenticando il sorgente appena visto e ogni informazione su *CRACKME01*, ci poniamo di fronte all'eseguibile come se fosse una scatola chiusa, totalmente sconosciuta. Il primo passo da fare è quello di iniziare l'analisi del file EXE usando uno scanner (ad esempio *Stud_PE*) e successivamente procedere a disassemblare il codice per studiarlo, ad esempio usando *BDASM*. L'analisi con *Stud_PE* rivela subito che il file è stato compilato attraverso MS Visual C++ versione 6.0 (vedi Fig. 1), quindi significa che si può intervenire direttamente con un disassemblatore per studiare il codice (non c'è nessun packer che protegge il file) e che il codice inizierà con alcune chiamate standard, inserite dal compilatore C++, alle API di sistema *GetVersion* e *GetCommandLineA*, seguite dal *main()* del programma. Si procede quindi con un'analisi più approfondita, realizzata mediante il disassemblatore *BDASM* (ricordiamo che questo programma richiede la libreria *MFC70.DLL*). L'analisi rivela subito

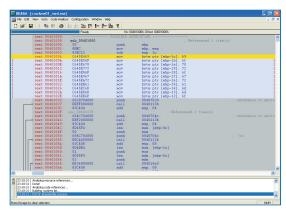



Fig. 6: Dopo la modifica al codice di *CRACKME01*, la password *"ioprogrammo"* non è più palesemente visibile all'interno del codice disassemblato. La password viene ora composta un carattere alla volta dalle istruzione *MOV BYTE PTR*.

(vedi Fig. 3) la presenza della stringa "ioprogrammo" in testa al codice (offset *0x401006*). Non ci vuole certo un mago del reverse engineering per capire che molto probabilmente questa stringa è proprio la licenza necessaria per l'esecuzione; una ulteriore conferma viene fuori anche dall'analisi fatta col decompilatore *REC*. *REC* è una utility a linea di comando da lanciare seguita dal nome del file eseguibile da decompilare. Ecco un estratto del file generato per *CRACKME01*:

```
L00401000() {
        /* unknown */ void  Vffffffec;
        /* unknown */ void  Vfffffff0;
        /* unknown */ void  Vfffffff4;
  Vffffffec = "ioprogrammo";
  L0040110B("\nTESO-WARE Versione 1.0\n\n");
  do {
    L0040110B("\nInserire una licenza valida : ");
    L004010F4("%s", & Vfffffff4);
    Vfffffff0 = L00401070( & Vfffffff4, Vffffffec);
  } while(Vfffffff0 != 0);
  L0040110B("\n\n:LICENZA VALIDA\n");
  return(L0040110B("\n\n:UTILIZZO DEL PROGRAMMA
CONSENTITO\n")); }
...
...... //continua
```

REC è stato in grado di ricostruire parte del main (identificato con la label *L00401000*) ed interpretando il listato prodotto, si capisce immediatamente che la variabile *Vfffffff4* letta da input viene confrontata con la variabile *Vffffffec*, che corrisponde alla stringa "ioprogrammo". Il risultato del confronto viene memorizzato nella variabile *Vfffffff0* che è quella che pilota l'intero ciclo di while che realizza il controllo della licenza. In questo primo semplice caso non è stato necessario ricorrere al debugger, ma è bastato analizzare i sorgenti decompilati del programma per capire come forzarlo; nella vita reale il reverse engineer deve fronteggiare protezioni ben più dure di queste e inoltre il listato prodotto da decompilatori e disassemblatori è spesso complicato da interpretare a causa della miriade di istruzioni e dalla presenza dell'interfaccia grafica.

RAFFORZIAMO LA PROTEZIONE

L'errore commesso (volutamente) in *CRACKME01* è quello di memorizzare la password di accesso in maniera visibile all'interno del file eseguibile. Una cosa del genere è l'antitesi della sicurezza! La procedura più sicura sarebbe quella di memorizzare nel file la chiave hash della password ed effettuare il confronto solo fra chiavi hash, ma non pretendiamo certo di implementare l'algoritmo *MD5* per questo semplice esempio, quindi ricorreremo ad un trucco più banale. Invece di salvare la password in maniera diretta e in una sola volta, proviamo a comporla carattere per carattere, usando un vettore di *char*, in questo modo:

Mettersi alla prova

Una volta acquisita dimestichezza col debugger e con gli altri tool di reverse engineering, si può tentare di aumentare la difficoltà della sfida e provare a forzare protezioni più sofisticate. Su Internet esistono una marea di siti dedicati al reverse engineering, dove appassionati e studiosi di ogni parte del mondo confrontano tecniche e scambiano informazioni. Molti di questi siti mettono a disposizione dei principianti una sezione di CRACKME di diversa difficoltà, scritti proprio per essere forzati da reverse engineer che vogliono misurare la propria abilità.

Sicurezza

Costruire
un crack per le applicazioni Windows

```
//modifica anti-disasm
/*
char pwd[11];
pwd[0]='i';
pwd[1]='o';
...
pwd[8]='m';
pwd[9]='m';
pwd[10]='o';
//----------------------------
*/
```

Naturalmente si potrebbe ancora migliorare il tutto invertendo ad esempio l'ordine delle assegnazioni dei caratteri (in modo da non avere la sequenza diretta "*ioprogrammo*" ma qualcosa tipo "*mmoargiorop*"), oppure memorizzando i codici ASCII successivi di ogni carattere ("*jpqsphsbnnp*" al posto di "*ioprogrammo*") per poi decrementarli di uno al momento del controllo. Per il momento però ci accontenteremo anche della prima semplice modifica. Come si vede dalla Fig. 6, le istruzioni di assegnazione del vettore sono state convertite nell'equivalente assembly *MOV BYTE PTR [EBP-xx]* che ha il vantaggio di non rivelare palesemente la stringa contenente la password agli occhi di chi usa il disassemblatore. In questo caso il reverse engineer è costretto a ricorrere all'analisi col debugger: useremo per l'occasione *OllyDbg*. Apriamo col debugger il file *CRACKEM01_MOD.EXE* ottenuto compilando la versione modificata di *CRACKME01* e iniziamo a tracciare col debugger le istruzioni passo dopo passo usando *F8 (step over)*. Dopo aver incontrato una chiamata a *GetVersion* ed una a *GetCommandLineA* (come era stato preannunciato), troviamo la chiamata al main del programma alla riga *0040121B*, preceduta da tre salvataggi nello stack (*PUSH*). Giunti in questo punto bisogna scendere all'interno della *CALL* usando *F7 (step into)*, che ci porta in questa zona del programma, che è l'inizio vero del nostro sorgente:

```
:00401000 55            push ebp
:00401001 8BEC          mov ebp, esp
:00401003 83EC1C        sub esp, 0000001C
:00401006 C645E469 mov [ebp-1C], 69 // "i"
:0040100A C645E56F mov [ebp-1B], 6F  // "o"
:0040100E C645E670 mov [ebp-1A], 70  // "p"
:00401012 C645E772 mov [ebp-19], 72  // "r"
:00401016 C645E86F mov [ebp-18], 6F // "o"
:0040101A C645E967 mov [ebp-17], 67  // "g"
:0040101E C645EA72 mov [ebp-16], 72  // "r"
:00401022 C645EB61 mov [ebp-15], 61  // "a"
:00401026 C645EC6D mov [ebp-14], 6D // "m"
:0040102A C645ED6D mov [ebp-13], 6D // "m"
:0040102E C645EE6F mov [ebp-12], 6F // "o"
:00401032 6830704000 push 00407030
                               //stringa "TESO-WARE..."
:00401037 E8FF000000 call 0040113B //printf
```

La subroutine inizia, naturalmente, con una *PUSH EBP*, necessaria per salvare la base dello stack; successivamente troviamo le istruzioni *MOV [EBP-xx]* che servono a memorizzare la password nel vettore un carattere per volta (i caratteri *69, 6F, 70*, ecc. sono quelli della stringa "*ioprogrammo*").
Proseguendo lo step over col comando F8, si incontrerà la chiamata a *printf (CALL 0040113B)* che visualizza una stringa su console e quindi si entrerà, all'offset *0040103F*, nella routine cruciale del programma.

```
:0040103F 684C704000 push 0040704C
                         //stringa "Inserire una lic.."
:00401044 E8F2000000 call 0040113B //printf
:00401049 83C404  add esp, 00000004
:0040104C 8D45F4  lea eax, dword ptr [ebp-0C]
                                 //offset stringa input
:0040104F 50                push eax
:00401050 686C704000 push 0040706C //stringa "%s"
:00401055 E8CA000000 call 00401124  //scanf
:0040105A 83C408               add esp, 00000008
:0040105D 8D4DE4  lea ecx, dword ptr [ebp-1C]
                                 //offset stringa input
:00401060 51 push ecx
:00401061 8D55F4 lea edx, dword ptr [ebp-0C]
                               //offset password vera
:00401064 52    push edx
:00401065 E836000000 call 004010A0 //strcmp
:0040106A 83C408 add esp, 00000008
:0040106D 8945F0 mov dword ptr [ebp-10], eax
:00401070 837DF000 cmp dword ptr [ebp-10],
                           00000000 //passord esatta?
:00401074 75C9 jne 0040103F //se no, ritenta
```

Il codice disassemblato appena mostrato salva una stringa da input (memorizzandola all'indirizzo [EBP-OC]) e infine la confronta, mediante una chiamata a strcmp, con la password reale. L'esito del confronto viene deciso subito dopo l'istruzione *CMP DWORD* al

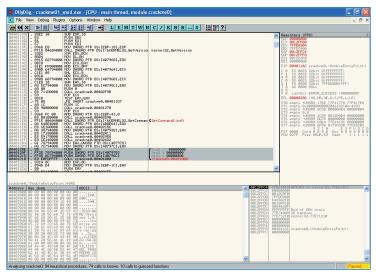



Fig. 7: *OllyDbg* in azione sul file *CRACKME01.EXE*. Il main di un programma C++ generalmente si trova subito dopo le chiamata a GetVersion e *GetCommandLineA*, preceduto da alcuni salvataggi (*PUSH*) nello stack.

rigo *00401070* e attraverso un salto condizionato *JNE* (salta se non è uguale), che fa ritornare l'esecuzione al punto di partenza *(0040103F)*. Modificando tale salto in *JE* (salta se uguale) o sostituendolo con delle istruzioni *NOP*, si forza il programma *CRACKME01*. La modifica può essere scritta mediante un hex editor (vedi Fig. 5), ricercando nel file *EXE* la stringa *"837DF00075C9"* e modificandone la parte finale con *"74C9"* (che equivale a *JE*) oppure con *"9090"* (che equivale a *NOP*).

Per proteggere ulteriormente *CRACKME01.EXE* da quest'altro attacco col debugger, si poteva pensare di comprimerlo con un packer (ad esempio *UPX*), che lo rende immune dalla decompilazione e che inoltre non consente la modifica diretta, tramite hex editor, dei bytes al suo interno (sono avvolti dal packer).

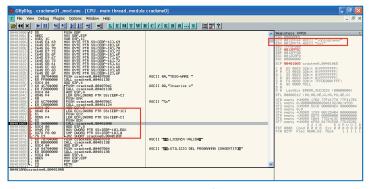



Fig. 8: **Al momento del confronto si può chiaramente vedere col debugger (particolare sui registri ECX e EDX) qual è la password reale (ioprogrammo) e quella immessa dall'utente. Il confronto è realizzato dall'istruzione CMP DWORD ed è seguito da un salto condizionato.**

E' scontato dire che su Internet sono comunque reperibili numerosi unpacker per *UPX*, capaci di riportare alla luce il *CRACKME01.EXE* originale.

CRACKME02... SEMPRE PIÙ DIFFICILE!

L'eseguibile scelto per CRACKME02 è un file tratto da *http://www.crackmes.de/crackmeinfo.php?ID=540* e, a differenza di *CRACKME01*, questa volta non disponiamo di alcun sorgente e in più ci troviamo di fronte ad un'applicazione dotata di interfaccia grafica… la faccenda inizia a complicarsi!

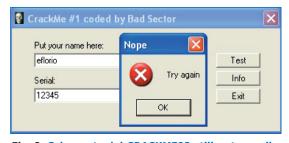



Fig. 9: **Schermata del *CRACKME02* utilizzato per il secondo esperimento. Si tratta di una applicazione a finestra, di conseguenza bisogna intercettare le chiamate ad API come *MessageBoxA* e *GetDlgItemTextA*.**

Eseguendo il file ci accorgiamo subito che *CRACKME02* richiede una coppia di valori *username/serial* che dovranno combaciare in base a qualche algoritmo o funzione matematica a noi sconosciuta. L'analisi con *Stud_PE* ci rivela che il file *EXE* è stato compilato con *MASM/TASM*: il lato buono è che non dovremo combattere contro un packer, quello cattivo è che l'autore, avendo scritto il programma in Assembly, può aver messo qualche trappola o delle routine molto intricate.

Dando un'occhiata alla finestra *Functions* notiamo che l'eseguibile importa alcune funzioni dalla libreria *USER32.DLL* del sistema, in particolare la nostra attenzione cade su *MessageBoxA* e su *GetDlgItemTextA*. Quando si immette un serial number errato, il programma mostra una finestra di dialogo *"Try again"*, ricorrendo sicuramente alla API *MessageBoxA*. Senza perderci di morale mettiamo quindi mano al debugger (questa volta useremo *SoftICE*) e impostiamo subito qualche breakpoint utile: attiviamo il debugger con *CLTR+D* e settiamo *"BPX MessageBoxA"*, che ci aiuterà sicuramente. Facendo scattare nuovamente la finestra *"Try Again"*, questa volta finiremo nel debugger, che intercetta la chiamata alla API. Premendo *F11* e quindi *F10* (per seguire la *RET*), usciremo dalla chiamata a *MessageBoxA* e torneremo nel punto del programma dove è avvenuta la *CALL*.

Sicuramente il controllo sull'esattezza del serial number avviene prima della chiamata a *MessageBoxA*, quindi scorriamo a ritroso il codice (*CTRL* + freccia su) fino a quando non ci imbattiamo in una *GetDlgItemTextA*, la API usata per prelevare l'input dall'utente su una finestra di Windows.

Conviene senz'altro mettere qui un altro breakpoint *(BPX 401148)* e far ripetere il controllo del serial number al *CRACKME*; questa volta il debugger interviene prima che venga mostrata la finestra *"Try Again"*, portandoci nel cuore della routine che genera il seriale: ricorrendo al disassemblatore possiamo analizzare il codice presente all'offset *401148*.

INVENTARSI UN KEYGEN

Nell'esempio del *CRACKME01* siamo ricorsi a un "cracking" brutale dell'applicazione, forzando il programma ad eseguirsi, qualunque fosse la password immessa. Questa volta cercheremo di essere più "garbati", giocando ad indovinare l'algoritmo di generazione del serial number.

Dando un'occhiata alla routine individuata prima *(offset 401148)*, si può notare che innanzitutto il seriale deve essere maggiore di 4 caratteri e che i calcoli fatti per generare il codice si basano su *EAX*, che contiene, in questo punto del programma, la lunghezza dell'username immesso dall'utente (*EAX* contiene il vaore restituito da *GetDlgItemTextA*).

```
:00401148 E8DE000000 Call 0040122B //GetDlgItemTextA
:0040114D 83F800 cmp eax, 00000000
```

Sul Web

REC, il decompilatore per linguaggio C

http://www.backerstreet.com/rec/recita.htm

CrackMe di ogni genere per esercitarsi col reverse engineering

http://www.crackmes.d
http://members.lycos.co.uk/sccssub/crackme.html

Siti dedicato al reverse engineering

http://www.lockless.com/tutorials.html
http://pmode.impazz.it/

```
:00401150 0F8499000000 je 004011EF
                                   //se lung.=0 seriale errato
:00401156 83F804  cmp eax, 00000004
                                   //se lung.<4 seriale errato
:00401159 0F8290000000 jb 004011EF
:0040115F 33C9              xor ecx, ecx
:00401161 33DB              xor ebx, ebx
:00401163 33F6              xor esi, esi
:00401165 8945FC        mov dword ptr [ebp-04], eax
//subroutine1
:00401168 0FBE81F3204000  movsx eax, byte ptr
                                              [ecx+004020F3]
:0040116F 83F820              cmp eax, 00000020
:00401172 7407                je 0040117B
:00401174 6BC004  imul eax, 00000004 //moltiplica x 4
:00401177 03D8 add ebx, eax //somma il valore ottenuto
:00401179 8BF3                mov esi, ebx
:0040117B 41                  inc ecx
:0040117C 3B4DFC  cmp ecx, dword ptr [ebp-04]
:0040117F 75E7         jne 00401168
:00401181 83FE00              cmp esi, 00000000
:00401184 7469                je 004011EF
//subroutine2
:00401186 BB89476500 mov ebx, 00654789
                                        // valore di partenza
:0040118B 0FBE81F2204000 movsx eax, byte ptr
                                              [ecx+004020F2]
:00401192 4B dec ebx //decrementa
:00401193 6BC302  imul eax, ebx, 00000002
                                          //moltiplica per 3
:00401196 03D8 add ebx, eax //somma il valore ottenuto
:00401198 4B dec ebx //decrementa
:00401199 49  dec ecx
:0040119A 75EF jne 0040118B
:0040119C 56                 push esi
:0040119D 53                 push ebx
* Possible StringData Ref from Data Obj ->"BS-%lX-%lu"
        |
:0040119E 68C7204000 push 004020C7
                                //il serial number inizia con
:004011A3 68BB214000  push 004021BB //la stringa "BS"
:004011A8 E86C000000  Call 00401219 //wsprintfA
:004011AD 58                 pop eax
:004011AE 58                 pop eax
:004011AF 58                 pop eax
:004011B0 58                 pop eax
:004011B1 E801000000 call 004011B7 //routine di controllo
:004011B6 C3                 ret
```

In questo codice si individuano facilmente due subroutine che eseguono numerosi calcoli ripetuti (si noti la presenza dei salti *JNE*): sono queste le routine che producono le due parti del serial number e funzionano grosso modo così (in pseudo-codice C++) :

```
//username = "eflorio"
//N = lunghezza username (es. "eflorio" => N = 7)
//subroutine1
X = 0;
while(i<7)
{
  tmp = char(username[i]) * 4;
  X = X + tmp;
}
//subroutine2
Y = 0x654789;
while(i<7)
{
  Y1 = Y-1;
  Y2 = (Y1 * 3) – 1;
  Y = Y2;
 }
```

Usando come input "eflorio" si otterrà le seguente generazione di codici per il valore *X*:

```
it1:  "e" tmp = 65*4 = 194     X = 0 + 194 = 194
it2:  "f"  tmp = 66*4 = 198    X = 194 + 198 = 32C
it3:  "l"  tmp = 6C*4 = 1B0    X = 32C + 1B0 = 4DC
it4: "o"  tmp = 6F*4 = 1BC     X = 4DC + 1BC = 698
it5: "r" tmp = 72*4 = 1C8    X = 698 + 1C8 = 860
it6: "i" tmp = 69*4 = 1A4     X = 860 + 1A4 = A04
it7: "o" tmp = 6F*4 = 1BC    X = A04 + 1BC = BC0
X=BC0 (in decimale 3008)
```

Mentre per il valore Y si otterrà :

```
it1:    Y1=654788          Y2=12FD697
it2:    Y1=12FD696         Y2=38F83C1
it3:    Y1=38F83C0         Y2=AAE8B3F
it4:    Y1=AAE8B3E         Y2=200BA1B9
it5:    Y1=200BA1B8        Y2=6022E527
it6:    Y1=6022E526        Y2=12068AF71
it7:    Y1=12068AF70       Y2=3613A0E4F
Y=613A0E4F (poiché i registri sono a 32-bit,
                       si considerano solo 8 bytes)
```

Fig. 10: **Alla fine si scopre che il serial number di *CRACKME02* è formato da tre parti: inizia con la sigla *BS* ed è seguito da due valori (*X* e *Y*) calcolati a partire dall'username fornito in input. L'algoritmo di calcolo determina il keygen.**

CONCLUSIONI

Finalmente siamo arrivati al temine delle nostre fatiche: il serial number ottenuto per l'username "eflorio" sarà della forma *"BS-Y-dec[X]"* ovvero *"BS-61 3A0 E4F-3008"*.

Ing. Elia Florio

Sicurezza

Costruire *un crack per le applicazioni Windows*

Nota dell'autore

Spero di aver mostrato, in maniera chiara ed esaustiva, una piccola parte del mondo del reverse engineering, che è molto vasto e ricco di argomenti interessanti. Nella vita reale i cracker incontrano protezioni di gran lunga più sofisticate di quelle viste in questo articolo; tuttavia bisogna sempre ricordare che ogni protezione, anche la più complicata, rimane sempre una creazione della mente umana e pertanto è imperfetta e non immune da errori. Nessuna protezione è sicura al 100% e la storia lo ha spesso dimostrato nel corso degli anni.

Delphi abbraccia la filosofia Microsoft .NET.

Delphi .NET: rivoluzione della specie

Il futuro di Delphi

Con l'uscita di Delphi 7, Borland ha distribuito una versione sperimentale del compilatore Delphi per la piattaforma .NET. Si tratta solo di una versione Preview, ma già denota i caratteri rivoluzionari di questa che non è solo l'ennesima versione del compilatore, ma un vero e proprio cambiamento di rotta.

Ad osservare la storia dell'evoluzione di Delphi ci si rende conto che il prodotto non è mai davvero cambiato, a dimostrazione di un ottimo progetto iniziale che non ha richiesto aggiustamenti significativi, ma anche, a mio avviso, di un minor impegno da parte del produttore. Certo è uscito Kylix, e Delphi, per la prima volta, si è aperto ad una piattaforma che non era Windows. Si trattava, però, quasi di un dettaglio implementativo: il compilatore produceva il solito codice macchina x86 e il framework, opportunamente riadattato per essere cross-platform, astraeva le differenze del sistema operativo. Non si poteva parlare di una novità epocale da un punto di vista tecnico, anche se vi era il principio di un sistema multi-piattaforma. Inoltre, già da diversi anni Borland ha tentato la strada del multilinguaggio sullo stesso IDE e, soprattutto, sullo stesso framework, con C++ Builder che altro non è che un'edizione C++ dell'ambiente Delphi: stesso framework di base, stessi sistemi operativi supportati (ovviamente grazie al framework che fa da mediatore) e diverso linguaggio.

VCL e CLX

VCL, Visual Component Library è il primo esempio di un framework estesissimo, formato da migliaia di classi, per di più fornite in sorgente (anche se con una licenza blindata) e capaci di "frustrare" ogni tentativo degli sviluppatori di scriversi librerie custom, per la semplice ragione che sicuramente il framework lo fa già... VCL ha la caratteristica di essere pensato e scritto su Win32 (prima Win16 con Delphi 1), per cui, quando Borland ha deciso il supporto ha Linux, ha scritto CLX che si poneva proprio l'obiettivo di essere multipiattaforma.

1994: L'AVVENTO DI JAVA

Java ha rappresentato una rivoluzione da numerosi punti di vista e ha avviato un'era nuova nella programmazione moderna: concetti come macchina virtuale, bytecode, metadati e ambiente di runtime protetto e verificabile probabilmente non erano del tutto nuovi, ma rappresentavano un'enfatizzazione estrema e mai vista in precedenza. Così, l'ambiente di esecuzione non è più il processore fisico e il sistema operativo, ma una macchina virtuale dotata di un assembly intermedio, cioè rappresenta una via di mezzo tra il codice sorgente di un linguaggio ad alto livello ed il linguaggio macchina dei microprocessori ed è in grado di gestire concetti ad alto livello come oggetti e metadati. Borland d'altro canto, non potendo più contare sulla capacità di offrirsi come ideatore e promotore di standard e sulla scorta del fatto di aver prodotto forse il miglior ambiente di sviluppo su tecnologia di altri (Delphi per Windows), decise di fare altrettanto con Java: produrre cioè il miglior ambiente di sviluppo Java, però basandosi sul compilatore che Sun distribuisce gratuitamente con il suo JDK. È nato così JBuilder che ha centrato l'obiettivo e tutt'ora viene considerato il miglior ambiente di sviluppo per Java.

1996-1997: HEJLSBERG E LA DIASPORA

Il fenomeno Java diventava sempre più forte, gli sforzi di Borland verso questa piattaforma ormai occupavano la quasi totalità delle risorse "buone" dell'azienda al punto che Borland si stava trasformando in una Java company. Al contempo, però, il mercato e soprattutto gli sviluppatori non Java, che non avevano un grande feeling con il linguaggio del chicco, ma che percepivano la potenza e la novità della piattaforma Java, spingevano sempre più verso la laicizzazione della piattaforma, soprattutto attraverso la richiesta di linguaggi di programmazione che producessero bytecode Java ma su un front-end sintattico differente. Così Borland, spinta dalla stessa comunità di sviluppatori Delphi, decise di intraprendere un progetto (segreto) per utilizzare il front-end Delphi per produrre bytecode. A capo del progetto c'era Hejlsberg che, con il suo team, riuscì ad arrivare anche ad uno stato avanzato di progettazione e prima implementazione. Ma quello era un periodo difficile per l'economia di Borland che non poteva permettersi grossi investimenti, in più c'erano problemi tecnici di non facile soluzione, essenzialmente derivanti dalla natura simbiotica e quasi imprescindibile del linguaggio Java dalla piattaforma Java: il potente strumento di Sun non era stato pensato per poter disaccoppiare i due elementi e questo rendeva molto complicato il lavoro di Hejlsberg.

Così il progetto fu accantonato.

MICROSOFT .NET

.NET, come Java, è costituito essenzialmente da una macchina virtuale che gira al di sopra del sistema operativo (attualmente solo Windows, ma potenzialmente si tratta di un modello abbastanza portabile). Questa macchina virtuale offre quindi un ambiente di runtime delle applicazioni che esegue non direttamente codice macchina, ma un codice assembly specifico della macchina virtuale. Questo codice assembly MSIL *(Microsoft Intermediate Language)* è un linguaggio compilato che si pone a livello intermedio tra un sorgente di un compilatore ad alto livello e il codice macchina di un processore moderno. Presenta inoltre una serie di caratteristiche di semplicità e quasi di "povertà", quali il fatto di non usare registri, ma di essere completamente "stack oriented", di non effettuare ottimizzazioni esasperate. La macchina virtuale non esegue direttamente questo codice binario intermedio, ma questo viene sottoposto ad una fase di compilazione al volo *(Jitting)* che lo trasforma in codice macchina ottimizzato per la macchina su cui sta girando. Un elemento fondamentale è l'ampio framework a disposizione di ogni applicazione .NET: migliaia di classi che risolvono la gran parte delle problematiche applicative. Gli sviluppatori Delphi sono già abituati alla sterminata VCL, ma il framework .NET rappresenta un passo in avanti in fatto di razionalizzazione e modernità nella scelta dei pattern di programmazione. Tutti questi elementi hanno consentito uno dei più grossi vantaggi di .NET su tutti i rivali, diretti o indiretti: l'indipendenza dal linguaggio. Microsoft ha fornito già da subito diversi linguaggi, ma soprattutto ha fornito delle linee guida per produrre nuovi linguaggi che generino codice IL e che sfruttino il type system e il framework di .NET. Questo è sufficiente perché, una volta prodotta una libreria in un qualsiasi linguaggio con queste caratteristiche, questa sia utilizzabile in binario da tutti gli altri linguaggi.

DELPHI .NET

Come tutti il linguaggi compatibili con il CLR, anche Delphi .NET Preview consente di interoperare pienamente con tutto ciò che è scritto per gli altri linguaggi, grazie al formato standard di definizione delle librerie binarie e alla presenza dei metadati che pubblicano le informazioni del codice di libreria, consentendo a tutti gli altri compilatori di sfruttarne le classi definite in essa, come se fossero nativamente scritte nel linguaggio da cui si stanno richiamando. Infatti, dopo la compilazione, si perde in qualche modo l'impronta del compilatore specifico perché ciò che viene prodotto è un assembly .NET, cioè un'unità eseguibile contenente codice intermedio MSIL e metadati. Finalmente i programmatori Delphi potranno attingere allo sterminato bacino di produttori di librerie e di componenti di terze parti per prodotti Microsoft con la possibilità di usarli in modo nativo. Dunque Delphi .NET sarà un linguaggio pienamente CLS *(Common Language Specification)* compliant e produrrà del codice verificabile a runtime, cioè sicuro e pienamente gestibile da CLR.

IL PRIMO ESEMPIO

A questo punto si rende necessario un primo esempio di applicazione Delphi .NET, una semplice *Console Application:*

```
Program MyDelphiNET.Esempio;
{$APPTYPE CONSOLE}
uses
  System, System.Collections;
Type  //definizione delle classi
  [Serializable]
  MyClass = class
  private
    FName : String;
    FNickName : String;
  protected
    property NickName : String read FNickName write
                                              FNickName;
  public
    Birthday : Datetime;
    property Name : String read FName write FName;
    function GetAge : Double; overload;
    function GetAge(ReferenceDate : Datetime) : Double;
                                              overload;
    constructor Create(name : String; birthday : Datetime);
  end;
  MyColl = class(Hashtable)
  private
    function get_Item(key : String) : MyClass;
    procedure set_Item(key : String; value : MyClass);
  public
    property Item[key : String] : MyClass read get_Item
                                          write set_Item;
    procedure Add(key : String; value : MyClass);
  end;
//implementazione delle classi
function MyColl.get_Item(key : String) : MyClass;
begin
  Result := MyClass(inherited get_Item(key));
end;
procedure MyColl.set_Item(key : String; value : MyClass);
begin
  inherited set_Item(key, value);
end;
procedure MyColl.Add(key : String; value : MyClass);
begin
  inherited Add(key, value);
end;
constructor MyClass.Create(name: String; birthday:
                                              Datetime);
begin
  inherited Create();
  FName := Name;
```

Il futuro di Delphi

Delphi .NET:
rivoluzione della specie

CLR

Come tutti il linguaggi compatibili con il CLR, anche Delphi .NET Preview consente di interoperare pienamente con tutto ciò che è scritto per gli altri linguaggi, grazie al formato standard di definizione delle librerie binarie e alla presenza dei metadati che pubblicano le informazioni del codice di libreria, consentendo a tutti gli altri compilatori di sfruttarne le classi definite in essa, come se fossero nativamente scritte nel linguaggio da cui si stanno richiamando.

Il futuro di Delphi

Delphi .NET: rivoluzione della specie

```
  Self.Birthday := birthday;
end;
function MyClass.GetAge: Double;
begin
  Result:= Datetime.Now.Subtract(Birthday).TotalDays / 365;
end;
function MyClass.GetAge(referenceDate: Datetime):
                                                   Double;
begin
  Result:= Datetime.Now.Subtract(referenceDate
                                       ).TotalDays / 365;
end;
//sezione di codice Main dell'applicazione
var
  myObj : MyClass;
  myCol : MyColl;
begin
  myObj := MyClass.Create('Ciro', Datetime.Parse(
                                          '30/08/1974'));
  myCol := MyColl.Create();
  myCol.Add('ciro', myObj);
  Console.WriteLine(myObj.Birthday.ToString());
  Console.WriteLine(myObj.GetAge.ToString());
  Console.Read();
end.
```

Le novità, in questo brevissimo programma, spiccano numerose. A partire proprio dalla keyword Program. Essa contiene un nome di programma strutturato con un punto "." tra due parole. In .NET ogni unità di codice è contenuta in una sezione detta *Namespace*. Un assembly, cioè un'unità elementare di compilazione (eseguibile o DLL) conterrà sempre uno o più Namespace che possono avere una struttura gerarchica: ogni livello di gerarchia è separato da un punto, così, nel caso specifico, avremo il Namespace root *MyDelphiNET* e sotto di esso il Namespace *Esempio*. Se nel programma fossero presenti delle Unit, ognuna di esse costituirebbe un Namespace che si posizionerebbe gerarchicamente sotto il Namespace definito da Program. Questo si ripercuote naturalmente anche nelle *Uses* che possono richiamare unit con struttura gerarchica, infatti la nostra applicazione usa le unit *System* e *System.Collection* anche se in questo caso è corretto definirli Namespace e non Unit. Ma non è tanto questo l'aspetto interessante da sottolineare, quanto il fatto che queste due unit non fanno parte della libreria Borland, ma sono proprio parte del framework .NET, quindi stiamo semplicemente usando il framework da Delphi! Non vi sarà sfuggito che la prima unit è *System*, ma non si tratta della cara vecchia System di *RTL*, ma del namespace *System* di .NET e più precisamente del core di .NET, cioè l'assembly fondamentale *MSCORLIB.DLL*, quello ciò che contiene la definizione degli elementi di base del framework. E la cara vecchia System che fine ha fatto? Semplice: non esiste quasi più perché con Delphi .NET non serve: o meglio, per ragioni di compatibilità, Borland ha definito il namespace *Borland.Delphi.System*, che fa solo da collante di compatibilità verso il codice e le classi Delphi tradizionali e quindi non per definire la base del framework come avviene con le versioni Win32 e Linux di Delphi. Una per tutte: le classi Delphi .NET discendono tutte da *System.Object* di .NET, come è necessario che sia per tutte le classi .NET e non dalla *TObject* di VCL. Infatti anche la nostra classe *MyClass* (che ha perso la T per ragioni di aderenza alla convenzione .NET), non discendendo esplicitamente da nessuna classe, discende da *System.Object*. Un'altra "stranezza" di definizione della classe *MyClass* è il costrutto *[Serializable]*. È semplicemente la convenzione sintattica per definire attributi di una classe o di un metodo: il nome tra quadre rappresenta infatti l'attributo da associare alla classe. Nello specifico l'attributo *Serializable* dona alla nostra classe la capacità di essere serializzata (in uno stream, su file o in qualsiasi altra forma). Il resto della classe non presenta novità, ma ritroviamo le caratteristiche di OOP avanzata tipica di Delphi: un costruttore parametrizzato, un overload sul metodo *GetAge*, una property pubblica ed un protetta. La seconda classe definita nell'applicazione è *MyColl*, essa discende dalla classe *System.Collections.Hashtable* del framework .NET e rappresenta una tipica collection alla Visual Basic, ma con algoritmi hash. In essa sono stati ridefiniti alcuni metodi (*Add* in particolare) per ottenere una collezione tipizzata di oggetti *MyClass*. Questa è una caratteristica molto potente che mancava nella VCL: le collection. Il framework .NET ne implementa di tutti i tipi e così gli sviluppatori Delphi potranno sbizzarrirsi e non limitarsi ad usare array o quelle veloci ma tristi *TList* oppure quella macchinosissima *TCollection* che tutto è tranne che una vera collection. Ma soffermiamoci adesso sul metodo:

```
function MyClass.GetAge(referenceDate: Datetime):Double;
begin
  Result:= Datetime.Now.Subtract(referenceDate
                                       ).TotalDays / 365;
end;
```

Il breve codice è sconcertante: viene invocato un metodo sul tipo scalare *Datetime*, cioè la *Now* che sappiamo restituire un *Datetime* contenente la data corrente. Per cui *Datetime* sembrerebbe essere un oggetto, una classe contenente il metodo statico *Now* che restituisce un *Datetime*. Come se non bastasse, *Datatime* possiede anche un metodo, questa volta di istanza, che è *Subtract* che accetta come parametro un altro Datetime. Esso restituisce un tipo *System.Timespan*, cioè un offset di tempo. Il *Timespan* è evidentemente a sua volta un oggetto perché espone la proprietà *TotalDays* che esprime l'offset di tempo sottoforma di numero di giorni che viene diviso per *365* e restituito come valore di ritorno del metodo. E scoprireste caratteristiche simili su *String*, su *Double* e su tutti gli altri tipi scalari del *Pascal*. Cosa significa? Semplice: in .NET anche i tipi scalari sono oggetti che discendono da *System.Object*. Questo consente loro di essere trattati tutti indistintamente in modo polimorfico come *Object* e questo è già un primo fon-

Attributo .NET

Un attributo per .NET è un concetto un po' particolare che probabilmente avrete incontrato in numerosi articoli su .NET: si tratta, in breve, di una informazione dichiarativa che indica al compilatore, al runtime e qualsiasi altra entità (compreso il codice che userà la classe) come interagire con la classe stessa.

damentale vantaggio. Inoltre, essendo oggetti, sono dotati di metodi statici e di istanza che permettono di gestire le caratteristiche di base del tipo. Perciò troverete che tutti gli scalari di natura numerica (interi, float, date) sono dotati di un metodo statico *Parse* in grado di parserizzare stringhe trasformandole in interi, float e date, che il *Datetime* possiede metodi per aggiungere e sottrarre date e così via. Ma l'aspetto più interessante è che tutti i tipi sono definiti dal framework, costituiscono cioè il CTS, per cui devono essere usati da tutti i linguaggi, magari adoperando alias sul nome del tipo come dimostra la tabella a fondo pagina.

LA PRIMA COMPILAZIONE

Adesso proviamo ad effettuare la compilazione dell'applicazione col compilatore a riga di comando di Delphi .NET Preview fornito nella directory *Bin* del compilatore Delphi .NET Preview:

```
dccil.exe Esempio.dpr
```

Otterremo l'esempio *Esempio.Exe* che, mandato in esecuzione, si comporterà come una normale Console Application. Proviamo quindi da mettere il naso nel binario disassemblandolo… Con Visual Studio .NET di Microsoft viene fornita la comoda utility *ILDASM* che svolge proprio questo compito. In Fig. 1) è possibile

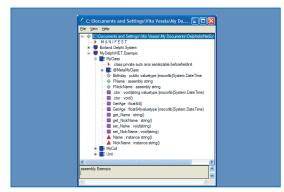



Fig. 1: **Il dissassembly del nostro programma.**

osservare il risultato del dissasembly del nostro assembly relativamente ai metadati pubblicati. Possiamo notare la nostra classe *MyClass* con la relativa definizione di metodi e proprietà, compreso il costruttore parametrizzato *(.ctor)*. è interessante notare che i tipi delle proprietà e dei parametri sono proprio i tipi di CTS. Adesso procediamo con un piccolo esperimento: riscriviamo la nostra applicazione in un altro linguaggio di .NET, cioè C#:

```
using System;
using System.Collections;
namespace ConsoleApplication1 { [Serializable]
  class MyClass { private string FName;
    private string FNickName;
    protected string NickName {
     get {return FNickName; }
     set {FNickName = value; } }
  public DateTime Birthday;
  public string Name { get {return FName; }
    set {FName = value; } }
public double GetAge() {
   return DateTime.Now.Subtract(Birthday).TotalDays / 365;}
public double GetAge(DateTime referenceDate) {
  return DateTime.Now.Subtract(
                    referenceDate).TotalDays / 365; }
public MyClass () { }
public MyClass (string name, DateTime birthday) {
  FName = name;
  Birthday = birthday; } }
class MyColl : Hashtable {
  public new MyClass this[string key] {
  get { return (MyClass)base[key]; }
  set { base[key] = value; } }
    public void Add(string key, MyClass value) {
      base.Add(key, value); } }
  class Class1 { [STAThread]
    static void Main(string[] args) {
       MyClass myObj = new MyClass("Ciro",
                        DateTime.Parse("30/08/1974"));
       MyColl myCol = new MyColl();
```

Tipo .NET	Tipo Delphi .NET	Tipo C#	Tipo Visual Basic .NET	Tipo JScript .NET	Tipi C++ con managed ext.
System.SByte	ShortInt	sbyte	SByte	byte	signed char
System.Byte	Byte	byte	Byte	byte	char
System.Int16	SmallInt	short	Short	short	short
System.UInt16	Word	ushort	-	-	unsigned short
System.Int32	Integer	int	Integer	int	int oppure long
System.UInt32	Cardinal oppure LongWord	Uint	-	-	unigned int oppure unsigned long
System.Int64	Int64	long	Long	long	__int64
System.UInt64	UInt64	ulong	-	-	unsigned __int64
System.Char	Char/WideChar	char	Char	char	wchar_t
System.Single	Single	float	Single	float	float
System.Double	Double/Extended	double	Double	double	double
System.Boolean	Boolean	bool	Boolean	bool	bool
System.String	WideString/string	string	String	String	String*

Tab. 1: **La for**

Il futuro di Delphi

Delphi .NET: *rivoluzione della specie*

Gli utilizzatori di C++ Builder

Ri, solo che non si tratta di una scelta davvero democratica, nel senso di permettere al programmatore di esprimersi in C++ anziché in Delphi, ma di un "adattamento" un po' forzato del linguaggio C++ al mondo Delphi, al punto che nessun programmatore C++ Builder può mai dimenticare, nemmeno per un istante, di lavorare con un modello ibrido basato su un framework Delphi e un compilatore C++ decisamente riadattato per l'Object Pascal; non a caso C++ Builder contiene al suo interno una versione del compilatore Delphi che è sempre allineata, se non più aggiornata a parità di numero di versione, a quella fornita con l'IDE di Delphi.

Il futuro di Delphi

Delphi .NET: rivoluzione della specie

Sul Web

Dove gli utenti Delphi 7 registrati potranno scaricare Delphi .NET Preview
http://www.borland.com/download

MSDN: la bibbia del programmatore Microsoft,
http://msdn.microsoft.com

Vito Vessia è cofondatore della codeBehind S.r.l.
http://www.codeBehind.it
una software factory di applicazioni enterprise, web e mobile, dove progetta e sviluppa applicazioni e framework in .NET, COM(+) e Delphi occupandosi degli aspetti architetturali. È autore del libro "Programmare il cellulare", Hoepli, 2002, sulla programmazione dei telefoni cellulari connessi al PC con protocollo standard AT+. Può essere contattato tramite e-mail all'indirizzo
vessia.pub@codeBehind.it

```
        myCol.Add("ciro", myObj);
        Console.WriteLine(myObj.Birthday.ToString());
        Console.WriteLine(myObj.GetAge().ToString());
        Console.Read(); }
} }
```

Fig. 2: Il dissassembly dell'omologo esempio scritto in C#.

Come avrete potuto osservare il codice è molto simile e, soprattutto, vengono invocati gli stessi metodi dei tipi del CTS, in particolare sulla classe Datetime. Non stupirà osservare quindi il dissassemblato dell'assembly prodotto da C# in Fig. 2: la classe *MyClass* è praticamente identica, a livello di metadati, a quella prodotta da Delphi .NET e questo dimostra che tutti gli assembly .NET sono uguali e si perde ogni riferimento al linguaggio che li ha generati! A questo punto, non vi lascerete impressionare nemmeno dalla Fig. 3 che mostra, a confronto, il codice IL del metodo GetAge prodotto dai due linguaggi. Ci sono alcune piccolissime differenza, soprattutto buone per quei giochi da settimana enigmista basati sul colpo d'occhio, ma nulla che lasci intendere al fatto che siano stati compilati a partire da un sorgente C# piuttosto che Delphi .NET.

Fig. 3: Il codice IL dei due programmi a confronto)

INTEROPERABILITÀ MULTI-LINGUAGGIO

Effettuamo una piccola modifica all'header e al footer del nostro programma di esempio e trasformiamolo in una DLL semplicemente sostituendo la keyword *Program* con:

```
Library MyDelphiNET.FirstNamespace;
```

Dunque eliminiamo tutto il codice della sezione *main*, che non ha più senso in una libreria ed effettuamo la compilazione. Otterremo l'assembly *Esempio.Dll*. Una sbirciatina con ILDASM ci rassicura sul fatto che si tratta proprio di un assembly/libreria .NET e non di una normale DLL Win32 e che espone le nostre due classi *MyClass* e *MyColl*. A questo punto scriviamo una semplice Console App in C# che sfrutta la nostra nuova libreria Delphi .NET. è sufficiente creare una nuova Console App e aggiungere un riferimento alla nostra Dll ripescandola con un semplice browsing del filesystem. Ecco il codice:

```
using System;
using MyDelphiNET.FirstNamespace;
namespace ConsoleApplication2 {
  class Class1 {[STAThread]
    static void Main(string[] args) {
      MyClass myObj = new MyClass("Ciro",
                              DateTime.Parse("30/08/1974"));
      MyColl myCol = new MyColl();
      myCol.Add("ciro", myObj);
      Console.WriteLine(myObj.Birthday.ToString());
      Console.WriteLine(myObj.GetAge().ToString());
      Console.Read(); }
}}
```

Con la *using*, corrispondente alla vecchia cara *Uses*, viene definito un alias al namespace *MyDelphiNET.FirstNamespace* che altro non è che il namespace pubblicato dalla nostra libreria Delphi. A questo punto si procede direttamente all'istanziazione e all'uso di *MyClass* e di *MyColl* senza avere nessuna cognizione del fatto che sia stata sviluppata in Delphi .NET: si tratta semplicemente di un assembly .NET richiamato da un altro assembly .NET!

CONCLUSIONI

Delphi .NET sarà un compilatore .NET di prima classe e lo dimostra già la versione Preview seppur ancora limitata e in fase di progettazione. Delphi .NET è .NET al 100%, è Delphi language al 90% perché alcune caratteristiche del linguaggio sono incompatibili con .NET e sarà Delphi VCL ad una percentuale molto più bassa nonostante l'obiettivo di Borland sia produrre VCL.NET che sarà uno strato soprastante .NET Framework ed in grado di essere parzialmente compatibile con la VCL di Delphi Win32. Quel che è certo, però, è che passare a Delphi .NET sarà una scelta di campo, ma probabilmente sarà una scelta senza rimpianti: sarà il passato che si fonde col presente quasi come se fossero opera di una stessa mano…

Vito Vessia

☑ Una libreria per la visualizzazione di grafici in Java.

Primi passi con JFreeChart

Grafica

In quest'articolo utilizzeremo, la libreria JFreeChart, per realizzare un'applicazione che visualizza una serie di grafici utili all'analisi di un portafoglio di investimenti finanziari.

Nelle applicazioni odierne, una corretta ed esaustiva visualizzazione di dati per via grafica è importante tanto quanto il loro collezionamento. Mostrare, ad esempio, i dati relativi al fatturato di un'azienda in un grafico ha il vantaggio di una maggiore facilità di analisi e di giudizio per chi li osserva.
Per lo sviluppatore, la creazione di grafici di qualità all'interno delle proprie applicazioni non rappresenta un'operazione facile e veloce, a meno che non utilizzi una libreria già pronta. *JFreeChart* è appunto una libreria di classi Java che consente la visualizzazione di grafici all'interno sia di applicazioni *stand-alone* basate su Swing che di applicazioni Web. Tramite le API della libreria, è possibile, facilmente e rapidamente, realizzare grafici professionali all'interno di applicazioni Java.
Nel corso di questo articolo, daremo un piccolo sguardo ai concetti di base della libreria mostrando altresì un piccolo esempio di applicazione utilizzabile da un consulente finanziario per analizzare il portafoglio di un cliente. L'applicazione consentirà di mostrare gli investimenti effettuati da un cliente, suddividendoli in base alle componenti azionario, obbligazionario e liquidità, permetterà inoltre di analizzare il rendimento ottenuto nel corso degli anni passati. Nelle figure sono mostrati i grafici generati dall'applicazione che andremo a realizzare.

Fig. 1: Composizione del portafoglio di un cliente

Fig. 2: Analisi dei rendimenti del portafoglio di un cliente nel corso degli anni.

LA LIBRERIA

JFreeChart è distribuito come free software direttamente dal suo sito (potete trovarlo anche nel CD allegato a ioProgrammo o nella sezione software di *ioProgrammo.net*) insieme al codice sorgente ed il suo uso è soggetto ai termini del *LGPL* (vedi relativo box).
La libreria mette a disposizione i file sorgenti e tutto l'occorrente per compilarli e per produrre la relativa documentazione in Java doc. E' presente un'applicazione dimostrativa (con relativo codice, ovviamente) che consiglio di osservare per scoprire molte delle potenzialità della libreria. E' presente inoltre un breve documento in PDF (*jfreechart-0.9.6-install.pdf*, nel caso della release *0.9.6*, utilizzata nel corso dell'articolo) che spiega sommariamente le caratteristiche della libreria, le modalità dell'installazione e la licenza d'uso. La documentazione aggiuntiva e/o tutorial sono a disposizione solo dietro pagamento. Mediante le API di *JFreeChart* possiamo generare diverse tipologie di grafici: lineari, a torta, a barre, time series, a step, Gantt, ecc… E' possibile inoltre combinare diversi grafici all'interno di un'unica area organizzandoli verticalmente od orizzontalmente. Caratteristiche aggiuntive, ma non per questo meno importanti, sono il tooltip, la gestione degli eventi del mouse, lo zoom, la possibilità di aggiungere delle note sul grafico e l'esportazione in

☑ CD ☑ WEB
Codice_JFreeChart.zip
cdrom.ioprogrammo.it

LGPL

L'acronimo LGPL sta per *Lesser General Public License*. Tale licenza si applica a particolari componenti software, tipicamente librerie.
La *General Public License* ribadisce la libertà di poter utilizzare e modificare il codice senza nessun costo, a patto di ridistribuirlo nella sua forma originale esattamente come lo si è ricevuto e completo delle eventuali modifiche apportate. La *LGPL*, fondamentalmente, si differenzia da questo tipo di licenza in quanto le applicazioni sviluppate, utilizzanti tale componente o libreria, non ereditano la caratteristica di free software (come invece avverrebbe nel caso della GPL).
Per ulteriori informazioni visitare il sito all'indirizzo http://www.fsf.org.

Grafica

Primi passi
con JFreeChart

vari formati quali JPEG, PNG, ecc.

CONCETTI DI BASE

Partiamo da alcuni concetti, necessari a capire le classi fondamentali della libreria. Gli oggetti creati nel momento in cui visualizziamo il grafico sono molteplici, ma alcuni di essi sono quelli più importanti. In particolare gli oggetti della classe *Plot*, che possiamo indicare come l'area in cui è mostrato il grafico. Si tratta di una classe astratta che si occupa di disegnare gli assi ed i dati e che definisce solo dei metodi generali; in quanto ogni tipologia di grafico usa una diversa classe concreta (ad esempio, nel caso dei grafici a torta, la classe concreta è il *PiePlot*). Ora concentriamoci sulla gestione dei dati, cioè le informazioni che vengono mostrate nei grafici. *JFreeChart* definisce delle interfacce (delle quali la principale è il *Dataset*) che permettono di astrarre le implementazioni delle strutture in cui vengono memorizzati i dati. In tal modo, si disaccoppia il codice relativo all'aggiornamento del grafico (*classe Plot*) da quello di gestione delle informazioni (*Dataset*). Tale meccanismo si basa sul *DatasetChangeListener* e fa sì che non appena aggiungiamo o rimuoviamo un valore nel *Dataset* notiamo un refresh sul grafico. Il *DatasetChangeListener* è un'interfaccia che identifica un'entità interessata ad essere notificata a fronte di una modifica nei dati (un esempio è appunto la classe *Plot*). Il *Dataset* si occupa invece di registrare, mediante i metodi *addChangeListener* e *removeChangeListener,* tali entità *DatasetChangeListener.* Il *Dataset* è soltanto l'interfaccia più generale. In realtà, ogni tipologia di grafico lavora con una o più interfacce dati derivanti da questa ed implementate da appropriate classi. Ad esempio, l'interfaccia *KeyedDataset* permette di organizzare i dati in coppie chiave-valore. I grafici a torta utilizzano, tra le altre, un'interfaccia derivante da questa, detta *PieDataset.* Entreremo in maggiore dettaglio man mano che svilupperemo la nostra applicazione. Per ora, concentriamoci sui principali componenti del grafico evidenziati in figura. In alto, sopra il *plot,* si può notare il titolo associato al grafico, mentre, al di sotto, è visibile la legenda dove sono riportati i nomi degli elementi grafici mostrati ed il relativo colore. La classe che stiamo osservando è quella più comune, la *StandardLegend* derivante da quella astratta, *Legend,* e che si differenzia per l'implementazione del metodo *draw*. I componenti descritti (e tutti gli altri che abbiamo tralasciato per semplicità) necessitano una sincronizzazione sia in fase di inizializzazione che in quella di aggiornamento. Per l'utente non sarebbe facile gestirli direttamente, in quanto dovrebbe rispettare la corretta sequenza delle diverse fasi di inzializzazione. La libreria ci viene in aiuto fornendoci la classe *JFreeChart* che contiene i riferimenti e gestisce tutti gli oggetti che vediamo nei nostri grafici: la legenda, il plot, il dataset, ecc. Ogni istanza rappresenta un grafico di un certo tipo e con certi dati. La creazione di un oggetto *JFreeChart* è possibile mediante la classe di utilità *ChartFactory.* Si tratta di una classe *factory* (vedi relativo box) che mette a disposizione dei metodi statici, i quali creano l'oggetto *JFreeChart* adatto, in base al tipo di grafico che desideriamo. Sono tali metodi che si occupano dell'effettiva inizializzazione di tutte le componenti. Infine, spendiamo alcune parole sull'integrazione con le Swing. A tal fine, la libreria mette a disposizione una classe derivante da *JPanel*, denominata *ChartPanel,* che agisce da contenitore di tutte le componenti che si occupano del disegno del grafico. Il pannello invocherà il metodo draw di questi ultimi passando l'oggetto *Graphics2D* ogni volta che necessita di ridisegnarsi. E' presente anche una classe *ChartFrame,* derivante da *JFrame,* che crea automaticamente al suo interno un pannello *ChartPanel* con il grafico desiderato.

Fig. 3: **Componenti fondamentali di un grafico.**

Classe Factory

Il concetto di classe *Factory* nasce dal pattern *Abstract Factory*, che offre un'interfaccia per la creazione di gruppi di oggetti in relazione tra loro senza specificare le loro classi concrete. Uno dei vantaggi di tale pattern è che permette di tenere sotto controllo le classi degli oggetti creati ed il punto in cui avviene la fase di inizializzazione di tali oggetti e delle loro relazioni. In tal modo, il codice cliente vede solo un gruppo di oggetti pronti ad essere utilizzati, senza curarsi delle implementazioni e della loro inizializzazione.

L'APPLICAZIONE

Dopo questi brevi concetti introduttivi, iniziamo a sviluppare l'applicazione di esempio. Come primo passo, costruiamo il grafico a torta che mostra la composizione del portafoglio del cliente. Esso può essere suddiviso, in base alla tipologia degli investimenti effettuati, nelle diverse componenti principali (azionario, obbligazionario, liquidità). Come esempio, utilizziamo i valori riportati in tabella.

Investimento	Valore %
Azionario	40,00
Obbligazionario	42,20
Liquidità	17,80

Abbiamo visto in precedenza che i grafici a torta utilizzano un tipo particolare di *Dataset*, il *PieDataset.* Le API ci forniscono un'implementazione di default con la classe *DefaultPieDataset* che utilizzeremo per gestire i nostri dati.

```
// create a dataset...
    DefaultPieDataset data = new DefaultPieDataset();
    data.setValue("Obbligazionario", 42.2);
    data.setValue("Liquidità", 17.8);
    data.setValue("Azionario", 40.0);
```

Grafica

Primi passi
con JFreeChart

Una volta creato l'oggetto che contiene i dati, è sufficiente invocare il metodo *setValue* per inserire una coppia chiave-valore nel *Dataset* rappresentante una porzione della torta. Una volta organizzati i dati, possiamo concentrarci sul grafico andando a creare l'oggetto *JFreeChart* mediante la factory.

```
// create a chart...
JFreeChart chart = ChartFactory.createPieChart(
        "Portafoglio cliente", data,true, true,false);
```

Per far ciò, sulla classe *ChartFactory*, invochiamo il metodo statico *createPieChart* che ci restituisce un oggetto *JFreeChart* con al suo interno un'istanza della classe concreta *PiePlot*. Il primo argomento, passato al metodo, rappresenta il titolo del grafico mentre il secondo è il riferimento all'oggetto *Dataset* creato. Gli ultimi tre argomenti sono dei boolean che servono ad attivare alcune caratteristiche opzionali del grafico, rispettivamente: la legenda, il tooltip e l'utilizzo del generatore di URL per le mappe di immagini. A questo punto, non ci rimane altro che integrare l'oggetto *JFreeChart* in un'applicazione Swing. Per semplicità utilizziamo la classe *ChartFrame*.

```
// create and display a frame...
ChartFrame frame = new ChartFrame(
        "Demo portafoglio", chart);
frame.pack();
frame.setVisible(true);
```

Creiamo un'istanza di *ChartFrame* passandole il titolo che vogliamo che compaia sulla *JFrame*, e l'oggetto *JFreeChart* che intendiamo visualizzare. In figura possiamo osservare il risultato.

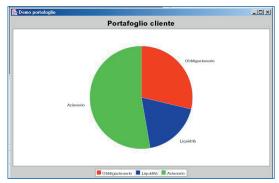



Fig. 4: Grafico portafoglio cliente Torta1.java.

Con pochi piccoli aggiustamenti, per esempio configurando alcuni parametri dell'oggetto *PiePlot*, possiamo cambiare sensibilmente l'aspetto del grafico. In ogni caso, *JFreeChart* permette di modificare molte delle proprietà del grafico (colori di sfondo, tipo di linee, ecc.) a runtime utilizzando il menù che appare quando si clicca sul grafico col pulsante destro del mouse. Per accedere alle proprietà del *plot* e modificarle direttamente nel codice, è presente sulla

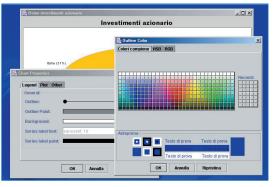
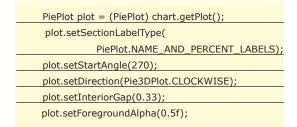



Fig. 5: Modifica proprietà grafico.

classe *JFreeChart* un metodo *getPlot* che ci restituisce il riferimento all'istanza del *Plot* creata. Ottenuto il riferimento, possiamo cambiare alcuni parametri di configurazione.

```
PiePlot plot = (PiePlot) chart.getPlot();
plot.setSectionLabelType(
        PiePlot.NAME_AND_PERCENT_LABELS);
plot.setStartAngle(270);
plot.setDirection(Pie3DPlot.CLOCKWISE);
plot.setInteriorGap(0.33);
plot.setForegroundAlpha(0.5f);
```

Mediante il metodo *setSectionLabelType*, configuriamo le informazioni che andranno scritte accanto ad ogni diversa porzione della torta. Possiamo decidere di visualizzare, singolarmente od in modo combinato, i nomi delle chiavi, i relativi valori o le percentuali. In questo caso siamo interessati a visualizzare sia i nomi sia le percentuali (per come abbiamo impostato i dati sono uguali ai valori). Un altro metodo, *setStartAngle*, ci permette di definire l'angolo iniziale dal quale disegnare le varie porzioni della torta (il valore di default è di 90 gradi). Con *setDirection*, decidiamo se le porzioni vengono disegnate in verso orario od antiorario, rispetto a come sono state inserite nel *Dataset*. Impostiamo la quantità di spazio vuoto lasciata intorno al grafico al 33% mediante *setInteriorGap*. Infine il metodo *setForegroundAlpha*, ereditato dalla classe generale Plot, imposta il livello di trasparenza del grafico. Come ultima modifica, rendiamo il grafico tridimensionale. Per far ciò, basta invocare il metodo *createPie3DChart* della *ChartFactory* con gli stessi parametri visti nel caso precedente. Il plot creato sarà, in questo caso, di tipo *Pie3DPlot*, una sottoclasse di *PiePlot*. A questo punto (il codice completo è quello della classe *Torta2*) otterremo quanto visto in Fig.1. Utilizzando ancora un *PiePlot*, possiamo realizzare facilmente un grafico che rappresenti ad esempio la suddivisione degli investimenti azionari del portafoglio del cliente. Possiamo, inoltre, estrarre una porzione della torta mediante il metodo *setExplodePercent* della classe *PiePlot*. Tale metodo richiede due parametri: l'indice (nel *Dataset*) della porzione da estrarre e la percen-

Gli assi

JFreeChart usa la convenzione di nominare l'asse relativo ai valori come range axis e l'asse relativo alle grandezze misurate come domain axis. Nel caso di dati organizzati in categorie, il domain axis corrisponde all'insieme delle possibili categorie mentre il range axis ai relativi valori. Invece per dati organizzati secondo coppie X-Y, domain e range axis corrispondono rispettivamente alla variabile indipendente e dipendente della funzione.
Gli assi sono gestiti tramite la classe astratta Axis e le sue sottoclassi. Queste vengono create all'interno dell'oggetto Plot in modo conforme con il tipo di plot durante la fase di inizializzazione effettuato dalla ChartFactory.

Grafica

Primi passi con JFreeChart

tuale di ingrandimento. Un esempio è mostrato nel codice della classe *Torta3* dove viene anche impostata una forma ellittica grazie al metodo *setCircular.*

```
plot.setCircular(false);
plot.setExplodePercent(1, 1.0);
```

Fig. 6: Investimenti azionari del portafoglio cliente (Torta3).

Passiamo ora ad un altro tipo di grafico, quello a barre. Esso risulta utile, nel nostro esempio, per mostrare un confronto tra i rendimenti ottenuti dal portafoglio del cliente nei vari trimestri degli anni passati dal 2000 al 2002 secondo quanto riportato in tabella.

	Gen-Mar	Apr-Giu	Lug-Set	Ott-Dic
2000	1,00	4,00	-3,00	-5,00
2001	-6,40	-5,34	-3,23	-1,23
2002	1,10	1,50	2,00	2,30

Come al solito, occorre effettuare la creazione dell'appropriato oggetto *JFreeChart*. La libreria offre diverse soluzioni per i grafici a barre: *2D* o *3D*, con orientamento verticale od orizzontale, ecc. Iniziamo partendo da un grafico a barre verticale in *3D:* sulla classe *ChartFactory* invochiamo quindi il metodo *createVerticalBarChart3D*.

```
JFreeChart chart = ChartFactory.createVerticalBarChart3D(
        "Rendimenti portafoglio", "Trimestre", "%",
        data, true, false, false);
```

Osserviamo subito che sono presenti due argomenti in più rispetto ai metodi di creazione dei grafici a torta. Tali nuovi argomenti rappresentano i titoli da assegnare agli assi del grafico in quanto, a differenza del grafico a torta, in questo caso sono presenti degli assi. Per saperne di più su come *JFreeChart* gestisce gli assi si consiglia di leggere il relativo box. Un'ulteriore differenza nei parametri riguarda il *Dataset*, che in questo caso è un *CategoryDataset*. Quest'ultimo permette di organizzare i dati secondo righe e colonne come riportato nella tabella vista in precedenza. In pratica, definiamo delle serie di dati (le righe) che rappresentano vari gruppi di valori misurati sulla base di alcune categorie (le colonne).

In questo caso, la classe concreta del plot che viene creato dal *ChartFactory* è di tipo *CategoryPlot*, specializzata nella visualizzazione di dati gestiti mediante il *CategoryDataset*. A questo punto possiamo impostare alcune caratteristiche della visualizzazione degli assi. Questi ultimi sono creati, in modo opportuno, dal costruttore della classe *Plot* e sono accessibili mediante i metodi *getDomainAxis* e *getRangeAxis*. Poiché abbiamo richiesto un grafico a barre orientato verticalmente, il plot creerà un *domain axis* orizzontale (mediante la classe concreta *HorizontalCategoryAxis*) che visualizzi le categorie dei dati ed un *range axis* verticale (mediante la classe concreta *NumberAxis*) che mostri i rispettivi valori in formato numerico. Possiamo quindi ottenere i riferimenti a questi oggetti e modificarne alcune proprietà. Ad esempio, per l'asse delle categorie, passando *true* al metodo *setSkipCategoryLabelsToFit,* impostiamo di non stampare i nomi delle categorie nel caso in cui ci sia il rischio che si sovrappongano se queste troppo ravvicinate.

```
HorizontalCategoryAxis domainAxis =
    (HorizontalCategoryAxis) plot.getDomainAxis();
domainAxis.setSkipCategoryLabelsToFit(true);
NumberAxis rangeAxis = (NumberAxis)
                              plot.getRangeAxis();
rangeAxis.setStandardTickUnits(
            NumberAxis.createIntegerTickUnits());
```

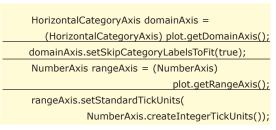
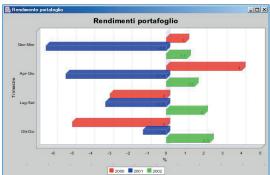



Fig. 7: Grafico a barre 3D orientato orizzontalmente.

Per l'asse dei valori, invece, impostiamo, mediante *setStandardTickUnits,* per le tacche di riferimento i valori delle unità. Eseguendo il codice riportato nella classe *Barre1*, otterremo la Fig. 2 vista in precedenza. In modo speculare, possiamo creare un grafico a barre orizzontale in *3D* (vedi codice della classe *Barre2*) o un semplice grafico a due dimensioni (vedi codice della classe *Barre3*). Abbiamo così visto, nel corso dell'articolo, come muovere i primi passi all'interno della libreria *JFreeChart*, mostrando la creazione di due tipologie di grafici: quello a torta e quello a barre. Consigliamo il lettore di sperimentare, agendo sui vari parametri dei grafici in modo da comprendere l'utilità e la funzione delle varie opzioni.

David Visicchio

Sul Web

http://www.object-refinery.com/jfreechart

Autore

David Visicchio è laureato in Ingegneria Informatica e lavora a Roma come designer /developer presso una multinazionale software, leader nel mercato per la persistenza ed il middleware di sistemi object-oriented. I suoi interessi sono orientati principalmente alle architetture distribuite basate su piattaforme J2EE e J2SE.

Un'applicazione per la gestione della posta elettronica.

Un mail client in Visual Basic (parte seconda)

Posta elettronica con VB

Nella prima parte, dedicata alla costruzione di un mail client in VB, abbiamo analizzato gli aspetti fondamentali che stanno alla base di un servizio di posta elettronica. In questa seconda ed ultima parte, vedremo come mettere in pratica quanto detto precedentemente, cercando di privilegiare gli aspetti più pratici rispetto a quelli puramente teorici.

Prima di vedere in dettaglio il progetto in Visual Basic, è bene fare alcune piccole considerazioni. La costruzione di un client di posta elettronica è certamente un'operazione molto complessa. Questo perché è necessario includere diverse funzionalità, che concorrono a rendere l'intera implementazione piuttosto complicata. Al fine di consentire un'agevole lettura del codice allegato e, soprattutto, allo scopo di permettere un rapido apprendimento delle tecniche e delle problematiche che devono essere affrontate, ci si è posti una serie di vincoli progettuali da soddisfare:

- Sviluppo del programma interamente in Visual Basic 6;
- Possibilità d'impostare la connessione predefinita di accesso remoto al fine di consentire la lettura e l'invio dei messaggi di posta elettronica sia attraverso rete sia via modem;
- Possibilità di agire sulle connessioni RAS presenti all'interno del PC, aggiungendone di nuove qualora necessario;
- Possibilità di registrare tutte le informazioni di configurazione, memorizzandole all'interno del registry di Windows, avendo peraltro la possibilità di rileggerle all'avvio dell'applicazione;
- Capacità di leggere mail con allegati da un qualunque server POP3;
- Capacità di inviare mail di solo testo con allegati;
- Logging delle principali informazioni relative a mail inviate e lette;
- Capacità d'iconizzare e gestire il programma all'interno della Tray Notification Area.

IL COMPONENTE WINSOCK

Quando si decide di sviluppare in VB un qualunque programma che debba colloquiare con un server, secondo le specifiche di un qualunque protocollo, l'oggetto che senza dubbio risulta esserci maggiormente d'aiuto, è il controllo *Winsock*. Esso è stato integrato in Visual Basic dalla Microsoft per gestire in maniera semplice e veloce le connessioni tra due PC mediante TCP/IP. Nello specifico, esso è sfruttato per realizzare applicazioni di tipo Client/Server tra un client di posta elettronica ed un server SMTP/POP3. Probabilmente voi tutti conoscerete questo tipo di oggetto ma, per dovere di completezza, credo sia opportuno premettere una sintesi delle principali proprietà e metodi. Innanzitutto è importante sottolineare che il controllo *Winsock* supporta due "modalità" di connessione, definite attraverso la proprietà *Protocol* del controllo stesso:

- **sckTCPProtocol:** modalità *TCP (Transmission Control Protocol);*
- **sckUBPProtocol:** modalità *UDP (User Datagram Protocol).*

In particolare, la prima modalità, consente di stabilire una connessione con un qualunque computer remoto facendo sì che due PC possano scambiare dati l'uno con l'altro sfruttando il protocollo TCP/IP come veicolo di trasporto per queste informazioni. Tra le proprietà coinvolte in quest'operazione le più importanti sono:

- **Proprietà RemoteHost**: nome del computer Server o indirizzo IP a cui collegarsi;
- **Proprietà RemotePort:** numero di porta TCP/IP alla quale collegarsi. Nel caso specifico di una connessione ad un server di posta elettronica, i possibili valori saranno *25 (SMTP)* e *110 (POP3);*
- **Proprietà State:** rappresenta lo stato del controllo *Winsock;*
- **Metodo Connect:** avvia la connessione al server secondo i parametri configurati precedentemente.

Winsock

Quando si decide di sviluppare in VB un qualunque programma che debba colloquiare con un server, secondo le specifiche di un qualunque protocollo, l'oggetto che senza dubbio risulta esserci maggiormente d'aiuto, è il controllo *Winsock*.

Posta elettronica
con VB

Un mail client
in Visual Basic

Quando la connessione è stata instaurata, entrambi i computer possono inviare e ricevere dati seguendo le specifiche imposte dal protocollo adottato (nel nostro caso SMTP o POP3).

- **Metodo SendData:** viene sfruttato per inviare i dati al server. In caso di esito positivo, ossia quando i dati sono stati ricevuti correttamente dal destinatario, viene generato l'evento *DataArrival*.
- **Metodo GetData:** questo metodo è invocato per recuperare i dati ricevuti. Esso va sfruttato all'interno dell'evento *DataArrival*.

Una generica connessione ad un server denominato *MyServer*, sulla porta TCP/IP PPPP, mediante un controllo *Winsock* denominato *WS1* potrebbe essere stabilita in maniera simile a:

```
' Definisce il protocollo da usare
WS1.Protocol= sckTCPProtocol
' Imposta i parametri essenziali al collegamento
WS1.RemoteHost="MyServer"
WS1.RemotePort=PPPP
' Connettiti al server
WS1.Connect
...
```

Stabilita finalmente la connessione con il server, può dunque avvenire lo scambio di informazioni.

LE COMPONENTI SOFTWARE DEL PROGRAMMA

Il progetto VB è formato da diverse form, da alcuni moduli di classe e da diversi moduli contenenti le dichiarazioni delle funzioni e delle costanti principali utilizzate all'interno del programma. All'interno del progetto sono anche stati definiti degli *User Control* che sostituiscono i comuni *Command Button* disponibili in VB. Le tecniche relative alla creazione e all'utilizzo di questi "oggetti" esulano dagli scopi di questo articolo, pertanto non saranno presi in considerazione. Di seguito è mostrato un elenco, suddiviso per soli *form* e *moduli*, che rappresenta tutte le componenti software del programma così come mostrato dalla lista disponibile all'interno dell'ambiente di lavoro di Visual Basic:

- **frmAbout:** questa finestra mostra alcune informazioni importanti circa il nome del programma ed i realizzatori;
- **frmConfig:** attraverso questa form vengono impostati tutti i parametri di configurazione utili al corretto funzionamento del programma. Da qui è possibile decidere l'account predefinito, la password d'accesso alla casella postale, il timeout del server, ecc;
- **frmLogo:** rappresenta la finestra di avvio del programma (*AKA splash screen*).
- **frmPOP3:** attraverso questa finestra è gestito il colloquio con il server di posta. Essa consente non solo di leggere la posta elettronica dal proprio account, ma anche di salvare gli allegati, inviare mail, configurare il programma stesso richiamando *frmConfig*, ecc. Rappresenta, in definitiva, la maschera principale di tutto l'applicativo;
- **frmSMTP:** la form *frmSMTP*, com'è facile dedurre, consente d'inviare una e-mail a uno o più destinatari, consentendo anche di gestire gli allegati;
- **frmTNA:** questa form costituisce il container per gestire correttamente la *Tray Notification Area;*
- **frmWinsock:** mostra le informazioni scambiate durante il colloquio tra il client ed il server di posta. Possiamo rilevare, in caso di problemi, quale sia la reale risposta del server a fronte di un nostro "comando".
- **CheckInternetConnectionModule:** in questo modulo sono contenute diverse funzioni che consentono di stabilire il tipo di connessione ad Internet attualmente in uso. L'utilizzo di un modulo siffatto è importante poiché è necessario ricordare che il collegamento ad Internet potrebbe essere attivo non solo attraverso un collegamento via modem, ma anche LAN/PROXY. In realtà alcune delle funzioni inserite in questo modulo sono qui presenti solo per maggiore completezza e praticità, anche se non sono sfruttate;
- **EffettiGrafici:** in questo modulo sono contenute tutte le funzioni grafiche che consentono di migliorare l'interfaccia del programma. Naturalmente la presenza di queste funzioni non è indispensabile per il corretto funzionamento del programma, anche se rendono l'applicazione più accattivante;
- **Encrypt:** questo modulo contiene la funzione che serve a criptare ed a decriptare una stringa. Nel nostro caso specifico essa è sfruttata semplicemente per "nascondere" la password d'accesso alla propria casella postale quando essa è memorizzata nel registry, ma potrebbe essere utilizzata anche per altri scopi come, ad esempio, conservare il serial number del programma;
- **Generale:** il modulo considerato contiene una serie di dichiarazioni di variabili e funzioni che servono per il corretto funzionamento del programma;
- **RAS_SMTP_POP:** com'è facile intuire, all'interno di questo modulo sono racchiuse tutte le funzioni e le dichiarazioni necessarie a gestire il collegamento ad Internet via modem e non, il colloquio secondo le specifiche SMTP e quello in POP3, ecc;
- **Registry:** qui dentro sono racchiuse tutte le funzioni che consentono il salvataggio e la corrispettiva lettura dei parametri di configurazione del programma all'interno del *registry*. Tramite le funzioni contenute in questo modulo, è possibile scrivere e leggere all'interno del *registry* di Windows tutti i parametri di configurazione del programma ed altre informazioni utili al corretto funzionamento dello stesso.

Chiunque di voi abbia deciso di dare un'occhiata al-

InfConf

La struttura *InfConf* è una struttura di tipo *Config* dichiarata all'interno del modulo *Generale.bas.* Il suo scopo è semplicemente quello di mantenere traccia dei parametri di configurazione essenziali al corretto funzionamento del programma.

l'applicativo, si sarà certamente accorto che tutto il codice risulta abbastanza commentato e che, in taluni casi, i commenti risultano essere in inglese anziché in italiano. Quest'anomalia è dovuta semplicemente al fatto che molte routine risultavano già preconfezionate e disponibili su Internet. Dopo questa piccola precisazione, possiamo passare ad una descrizione maggiormente dettagliata dell'intero progetto, tralasciando per semplicità aspetti ovvii o non strettamente legati all'implementazione di un client di posta. Per consentire di rendere il discorso maggiormente chiaro, tale descrizione sarà suddivisa nelle seguenti parti:

- La configurazione del programma;
- L'accesso ad Internet;
- La lettura dei messaggi di posta elettronica;
- L'invio dei messaggi di posta elettronica;
- La gestione degli allegati.

LA CONFIGURAZIONE DEL PROGRAMMA

Come abbiamo avuto modo di anticipare uno dei vincoli progettuali imposti durante la stesura del codice, è stato consentire il salvataggio delle informazioni essenziali al corretto funzionamento del programma, nonché il relativo ripristino al successivo avvio. Per poter realizzare questa funzionalità, potevano essere scelte diverse strade, una delle quali poteva essere quella di mantenere costantemente aggiornato un file *.INI* in grado di contenere tutte le informazioni necessarie. Per rendere le cose maggiormente "complicate", ma allo stesso tempo più interessanti, si è deciso di perseguire un'altra via: sfruttare il *registry* di Windows come container di questi dati. Il modulo *Registry*, descritto sommariamente nel paragrafo precedente, contiene la definizione di tutte le funzioni e delle variabili necessarie che concorrono al raggiungimento di questo obiettivo. In particolare, all'interno di questo modulo sono definite diverse funzioni tra le quali (elenchiamo solo le principali):

- **CreateNewKey:** Crea una nuova chiave all'interno del *Registry*.
- **SetKeyValue:** Imposta un valore in una chiave nel *Registry*.
- **QueryValue:** Restituisce il valore di una chiave del *Registry*.
- **Leggi_Registry:** Legge i valori dal *Registry* e li mette nella struttura *InfConf*.
- **Scrivi_Registry:** Scrive *InfConf* nel *Registry*.

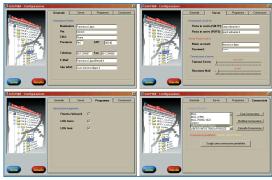



Fig. 1: La form frmConfig per la configurazione del programma.

La struttura *InfConf* struttura, ovviamente, è valorizzata all'avvio ed ogni qualvolta se ne abbia necessità. I valori di ogni elemento della struttura, come già detto, sono prelevati direttamente dal registry. La maschera *frmConfig* rappresenta l'unico modo per poter configurare tutti i parametri che ci interessano, a meno, ovviamente, di modificarli direttamente tramite l'utility Windows *regedit.exe*. Al suo avvio, l'evento *Load* della form scatena il seguente codice:

```
Private Sub Form_Load()
    Dim Entries(RAS_MaxEntryName) As RASENTRYNAME
    Dim NumConns As Integer
    Dim i As Byte
  ' Leggi le impostazioni precedenti dal Registry,
  ' mettendole in InfConf
    Leggi_Registry
  ' Aggiorna i campi con i valori di InfConf
    ReadConfig
  ' Aggiorna le 2 label relative alle 2 slider della finestra
    lblTimeoutSRV = TimeoutServer.Value & " secondi"
    lblRiczMail = IntervalRCV.Value & " minuti"
  ' Caricamento Accessi Remoti disponibili
    NumConns = GetRasConnections(Entries)
    If NumConns > 0 Then
      For i = 0 To NumConns - 1
          bbb = StrConv(Entries(i).szEntryName, vbUnicode)
          List1.AddItem Left(bbb, InStr(bbb, Chr(0)) - 1)
      Next i
    End If
End Sub
```

La funzione *Leggi_Registry* si serve di alcune API di Windows per leggere i dati dall'interno del registry, aggiornando conseguentemente la struttura *InfConf* affinché rifletta i dati di configurazione. Successivamente, viene lanciata la procedura *Read_Config* che aggiorna i vari campi della form *frmConfig* con i valori di quest'ultima struttura. Altro compito dell'evento *Load* della form è quello di preoccuparsi di "enumerare" tutte le *RAS Connection* definite all'interno del PC client. Anche per questo è sfruttata una funzione costruita ad hoc, *GetRasConnections()*, che sfruttando l'*API RasEnumEntries* ci consente di ottenere anche questo dato.

L'ACCESSO AD INTERNET

Affinché si possa scambiare informazioni con un server di posta elettronica su Internet, è necessario ovviamente essere connessi ad esso. Il progetto in VB è composto da un modulo, in particolare, in grado di rilevare se si è connessi ad Internet oppure se è necessario at-

Posta elettronica
con VB

Un mail client
in Visual Basic

SMTP e POP3

Una volta configurato il programma con i parametri necessari alla ricezione ed all'invio delle nostre mail, è possibile "colloquiare" con il nostro server di posta elettronica, comunicando con esso attraverso le specifiche SMTP e POP3.

Posta elettronica
con VB

Un mail client
in Visual Basic

tivare una connessione. Quando tentiamo d'inviare o di ricevere un'e-mail è necessario stabilire il "contatto" con il server in oggetto e tale scopo è esplicato dalla funzione *ConnectSRV()*. Essa risulta così strutturata:

```
Public Function ConnectSRV(WS1 As Winsock,
                    Host As String, Port As Integer) As Boolean
On Error Resume Next
  Dim Scelta As Integer  ' Determina la scelta
                         dell'utente, se continuare o abortire
   Dim ConnectionID As Long    ' Determina l'avvenuta
                                  connessione ad internet
   Dim DWFLAGS As Long     ' Flag utile a determinare
                                    il tipo di connessione
   Dim ConnessioneAttiva As Boolean    ' Determina se
                        è attiva una connessione ad Internet
   ConnectSRV = False
   Scelta = vbYes
  ' Intanto chiudi il socket...
   WS1.Close
  ' Imposta i parametri del collegamento con il server
   WS1.RemoteHost = Host
   WS1.RemotePort = Port
   WS1.LocalPort = 0
   ConnessioneAttiva = IsConnected()
   If ConnessioneAttiva Then
         WS1.Connect
         If WaitFor("", RispostaPREC) Then
            ConnectSRV = True
            Exit Function
         End If
   Else
      ' Tenta con il modem...
   While ConnectionID = 0
      ' Tenta il collegamento con Internet...
     ' Il valore di ritorno è restituito in ConnectionID
     InternetDial 0&, InfConf.ConnPredef,
         Iternet_Autodial_Fforce_Online, ConnectionID, 0&
        ' Ti sei collegato alla linea telefonica? Se NO...
        If ConnectionID = 0 Then
           Scelta = MsgBox("Errore in fase di
              collegamento!" + vbCrLf + "Vuoi ritentare?",
                                  vbYesNo, "Errore!")
             If Scelta = vbNo Then Exit Function
         End If
           Wend
           ' Ti sei collegato alla linea telefonica...
           If ConnectionID <> 0 Then
              WS1.Connect
              If WaitFor("", RispostaPREC) Then
                 ConnectSRV = True
                 Exit Function
              End If
           End If
  End If
End Function
```

In sostanza il suo funzionamento è molto semplice. Le vengono passati i tre parametri fondamentali ossia il controllo *Winsock* che dovrà gestire il colloquio con il server (riposto rispettivamente sulle form *frmSMTP* e *frmPOP3*), l'host e la porta di riferimento. Ottenuti questi parametri, *ConnectSRV()* imposta le relative proprietà del controllo *Winsock* passato come parametro e controlla lo stato della connessione attiva. Se non si dovesse essere connessi ad Internet, ad esempio, tenta di stabilire tale connessione attraverso l'avvio di una sessione remota attraverso modem, altrimenti dichiara aperto il collegamento con l'host remoto. Nel primo caso, in particolare, si serve della funzione *InternetDial()* che, opportunamente chiamata, consente di avviare la connessione RAS impostata come default. La presenza della funzione *WaitFor()*, usata peraltro moltissimo durante l'invio e la ricezione dei comandi in *SMTP/POP3*, garantisce il giusto tempo di attesa prima di dichiarare persa la connessione. L'omissione di questo "ritardo" significherebbe necessariamente un tempo di risposta immediato da parte del server, una situazione praticamente impossibile da verificarsi. Stabilita dunque la connessione con il nostro server, si può dare inizio allo scambio d'informazioni tra i due PC.

LA LETTURA DEI MESSAGGI DI POSTA ELETTRONICA

Il form principale, *frmPOP3*, rappresenta il punto d'inizio dal quale è possibile compiere tutte le principali azioni dell'applicativo, ivi compresa l'azione di lettura della posta elettronica. Esso è composto semplicemente da un menu principale, da una serie di pulsanti e da un paio di finestre che mostrano rispettivamente i dati principali delle mail ricevute ed il body di ognuna di esse. La lettura dei messaggi dal server di posta avviene mediante la pressione del pulsante *Ricevi*. Nel momento in cui esso è premuto, ha inizio il colloquio con il server POP3. La gestione del colloquio tra client e server è tutto centrato sull'utilizzo di tre funzioni, costruite ad hoc, che riflettono i tre stati coinvolti durante lo scambio d'informazioni in POP3:

- **AutenticaPOP3:** questa funzione altro non fa che inviare al server POP3 i comandi *USER* e *PASS* seguiti ciascuno dai parametri presenti in *InfConf*. Se tutto è stato scritto correttamente, il server ritorna una risposta positiva e si può procedere alla lettura dei propri messaggi;
- **TransactionPOP3:** durante questa fase vengono lette tutte le mail dalla propria casella di posta, nonché gli allegati di ognuna (argomento di un paragrafo successivo) mostrando l'elenco delle principali proprietà di ognuno di essi all'interno della griglia riposta su *frmPOP3*. Durante questa fase, in particolare, è aggiornata una struttura denominata *InfoARR* che tiene traccia delle principali caratteristiche di ogni mail scaricata. Essa, passata come unico parametro alla funzione *AggiornaDB()*, consente di popolare la griglia delle mail arrivate con le informazioni principali. Questa griglia, in realtà,

Allegati
Un file binario lo potremmo definire come una sequenza di caratteri "prelevati" dall'intero set dei caratteri ASCII.

fa riferimento ad un database MS Access contenente tutte queste informazioni. La struttura di questo DB, denominato *MailArrivate.mdb*, è molto semplice e può essere modificata secondo le esigenze.
Durante questa fase, inoltre, è aggiornato il file di log (attraverso una chiamata alla funzione *AggiornaLOG()*) con le informazioni più importanti circa ogni mail letta. La funzione *AggiornaLOG()* accetta in ingresso un solo parametro, identificato da due stringhe ben precise, *LOG_ARRIVI* e *LOG_ INVII*, che le indicano il tipo di log da effettuare. Al termine di questa fase, viene lanciato il comando *DELE* per contrassegnare ogni mail per la cancellazione.

- **UpgradePOP3:** questa funzione, la più semplice fa le tre esposte, non fa altro che chiudere la comunicazione con il server POP3, inviando semplicemente il comando *QUIT*. Durante questa fase, tutti i messaggi che risultavano contrassegnati con *DELE* durante la *Transaction Phase*, verranno definitivamente eliminati dalla casella postale.

Fig. 2: **La form frmPOP3 per la lettura e l'invio dei messaggi.**

L'INVIO DEI MESSAGGI DI POSTA ELETTRONICA

L'accesso alle funzionalità d'invio di un messaggio di posta elettronica avviene sempre attraverso la form *frmPOP3*, servendosi del pulsante *Invia*. A seguito della pressione di questo pulsante è aperta una nuova finestra, *frmSMTP*, dalla quale è possibile inserire i destinatari delle proprie mail, gli allegati ed il testo del messaggio. La funzione principale richiamata, dall'interno di questa form, è *InviaMail()* che non fa altro che stabilire il colloquio con il server SMTP di riferimento e invia l'e-mail ai destinatari. Alla stregua della ricezione dei messaggi di posta, si serve della funzione *ConnectSRV()* per stabilire la connessione con esso.
Tuttavia, è bene sottolineare un dettaglio importante, ossia il caso dell'invio di mail a più destinatari. In quest'ipotesi, infatti, è necessario scomporre la stringa che rappresenta questo dato, facendo bene attenzione a separare correttamente ogni indirizzo di posta elettronica con un ';'. Per estrarre correttamente ogni mail address, è stata predisposta una funzione costruita ad hoc, *ScomponiInvia()*, che si occupa di suddividere l'intera lista dei destinatari in tutti gli indirizzi e-mail, inviandoli "direttamente" al server SMTP durante il colloquio. Essa, infatti, è richiamata ogni qualvolta è necessario stabilire il destinatario di un'e-mail durante la fase di colloquio con il server SMTP ed è strutturata così:

```
Public Sub ScomponiInvia(ByVal Destinatario As String)
Dim LenDestMail As Integer
Dim NewPos, Pos As Integer
Dim DestinatarioMail As String
LenDestMail = Len(Destinatario)
frmSMTP.ProgressBar1.Value = 45
' Scomponi la lista dei destinatari
   For NewPos = 1 To LenDestMail
      Pos = InStr(NewPos, Destinatario, ";")
      ' Nessuna corrispondenza trovata...
       If Pos = 0 Then
          DestinatarioMail = Trim(Mid$(Destinatario,
                          NewPos, LenDestMail - NewPos + 1))
          NewPos = LenDestMail
       Else
          DestinatarioMail = Trim(Mid$(Destinatario,
                                     NewPos, Pos - NewPos))
          NewPos = Pos
       End If
      ' Invia la mail al destinatario corrente
         frmSMTP.WS1.SendData "RCPT TO:
                          <" & DestinatarioMail & ">" + vbCrLf
       If Not WaitFor("250", RispostaPREC) Then
          ' Se la TNA non è abilitata...,
                                    puoi mostrare le msgbox
           If frmPOP3.TmrRicez.Enabled = False Then
                            MsgBox "Errore in fase RCPT TO..."
          frmSMTP.WS1.Close
          Exit Sub
       End If
   Next
 End
End Sub
```

L'unico parametro passato alla funzione è la stringa che occorre suddividere, vale a dire il contenuto delle textbox *txtDestSMTP* e *txtDestCCSMTP* della form *frmSMTP*.

CONCLUSIONI

Come abbiamo potuto vedere, la creazione di un programma di tipo client/server come quello mostrato, spesso offre lo spunto per "ripassare" molti concetti di programmazione e non. In particolare, il progetto esposto, è stato proposto proprio con lo scopo di suscitare l'interesse di ognuno di voi a scoprire cosa si nasconda realmente dietro un programma che deve gestire un colloquio di tipo client/server con qualunque protocollo.

Alberto Lippo e Francesco Lippo

Posta elettronica
con VB

Un mail client
in Visual Basic

Possibili sviluppi

L'applicativo proposto può essere la base per un progetto di più ampio respiro. Di seguito, alcuni suggerimenti:

- **Possibilità di gestione di più account di posta elettronica;**
- **Gestione di MIME;**
- **Possibilità di mostrare un elenco delle mail arrivate, inviate, eliminate, ecc. alla stessa maniera offerta dai più diffusi client di posta elettronica come Outlook;**
- **Possibilità di gestire le risposte ad un e-mail, precompilando i campi necessari all'avvio della form *frmSMTP*.**
- **Configurazione di regole (filtri) per rigettare o archiviare in cartelle personali i vari messaggi di posta elettronica.**

Fatti e misfatti di RPC

RPC, DCOM e la porta 135 sono gli ingredienti alla base di una micidiale vulnerabilità scoperta dal team LSD e relativa alla famiglia di sistemi operativi targati Microsoft. Un bug giudicato di livello critico da tutti gli esperti.

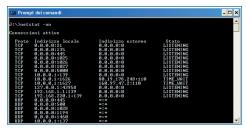



Fig. 1: **Netstat è il comando di Windows che rivela le porte aperte sul sistema, i sistemi operativi Microsoft aprono per default alcune porte standard che sono le 137/139 (NetBIOS), la 445 e la 135, quest'ultima relativa al servizio RPC interessato dal bug in questione.**

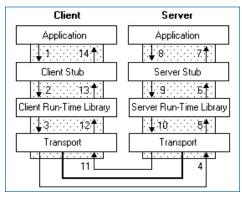



Fig. 2: **Schema logico di funzionamento del meccanismo di Remote Procedure Call (RPC) di Microsoft. A livello applicativo i dettagli dei layer di trasporto e di run-time vengono gestiti dal sistema operativo e sono trasparenti rispetto all'uso fatto dal programmatore.**

L'*unico computer sicuro è un computer spento*. Si tratta di un detto molto comune nel mondo hacker che, a prima vista, potrebbe sembrare un'esasperazione o un semplice paradosso, ma che – purtroppo – spesso viene avvalorato da alcune tristi vicende legate al mondo della sicurezza.

Non tutti hanno un'idea precisa di quante siano le porte aperte in ascolto (LISTENING) sul proprio sistema Windows mentre si è connessi a Internet. Nulla di più semplice da verificare, senza bisogno di essere un hacker, basta lanciare *"netstat –an"* (Fig. 1) e iniziare a dare i numeri : 135, 139, 445, 5000…e chi più ne ha, più ne metta! E non bisogna certo scomodare un esperto di sicurezza per chiarire il concetto di *"porta"*; non me ne vogliano i teorici dell'ISO/ OSI e del TCP/IP, ma brutalmente il concetto di "porta aperta" si definisce da sé: una porta rappresento un ingresso, un punto di passaggio attraverso il quale entrano ed escono dati da un sistema.

Ciò che un utente normalmente crede, in maniera inconscia, è che le informazioni in uscita siano soltanto quelle necessarie (….e qui ci sarebbe un lungo discorso da fare riguardo agli spyware) e allo stesso tempo si augura che nessun byte o pacchetto "malefico" (virus, worm, attacchi DoS e altre porcherie simili) riesca ad entrare senza autorizzazione.

Se la memoria non m'inganna già qualche tempo fa la porta 5000 di Windows (servizio *UPnP*), fu al centro delle "cronache mondane" della sicurezza a causa di un bug ritenuto pericoloso (vedi *ioProgrammo nr. 54*), per non parlare delle porte NetBIOS (137, 39 e 445) che da tempo affliggono i poveri amministratori di rete a causa delle loro insicurezze intrinseche (vedi *ioProgrammo n. 62*). E cosa dire delle altre porte?

C'è tempo per ogni cosa, basta solo avere pazienza: oggi, infatti, è il turno della porta 135 e del servizio RPC… e la prossima volta, a chi toccherà?

MICROSOFT E IL BOLLETTINO 03/26

La storia ha inizio il 17 luglio, quando il gruppo di ricerca LSD (*Last Stage of Delirium – http://lsd-pl.net*) pubblica su Bugtraq un advisory di sicurezza ritenuto di grado critico per l'intera famiglia di sistemi operativi Microsoft: NT, 2000, XP e addirittura il nuovissimo 2003 Server!

Nel documento, confermato peraltro anche da un bollettino di sicurezza di Microsoft (MS03-26) uscito quasi contemporaneamente, viene messo in evidenza come, attraverso la porta 135, aperta per default su tutti i sistemi MS, un ipotetico aggressore sarebbe in grado di eseguire codice su un host remoto, sfruttando un buffer overflow riscontrato dai "ra-

gazzi" del team LSD nell'interfaccia DCOM, relativa al servizio RPC di sistema.
Disorientati dalle sigle criptiche di Microsoft? Confusi sul significato delle diverse tecnologie?
Niente paura, ecco qualche piccola definizione che, senz'altro, sarà d'aiuto nella comprensione di alcuni dettagli tecnici di questo articolo:

- **RPC (Remote Procedure Call)**
  In base alla definizione classica, l'RPC è quel meccanismo usato per realizzare l'IPC ovvero l'*inter-process communication*. Si tratta di fornire ai programmi la capacità di scambiare dati e invocare metodi o funzioni che risiedono in un processo diverso rispetto a quello invocante.
  La potenza di questo meccanismo sta nel fatto che viene gestito in maniera trasparente (Fig. 2) rispetto al programmatore e che, inoltre, le invocazioni di metodi e procedure possono coinvolgere anche processi che risiedono su host diversi, connessi in rete locale o via Internet. In ambiente Windows il tutto si basa su un servizio chiamato *"endpoint mapper"* (visibile col nome di *epmap*), attivo proprio sulla porta 135, in attesa di connessioni da parte dei client RPC.

  RIF: *http://msdn.microsoft.com/library/en-us/rpc/rpc/microsoft_rpc_model.asp*

- **DCOM (Distributed Component Object Model)**
  Trattasi di un protocollo, originariamente nato come *"Network OLE"*, progettato per consentire ai componenti software di comunicare in maniera diretta attraverso le reti. In sostanza è una variante della tecnologia COM di Microsoft orientata per l'uso su network. Per realizzare alcuni servizi si avvale, a sua volta, del meccanismo fornito dall'RPC.
  È gestito nel sistema attraverso la console *\WINDOWS\SYSTEM32\DCOMCFG.EXE* (Fig. 3).

  http://www.microsoft.com/com/tech/dcom.asp

PANICO E CONFUSIONE IN RETE

Tornando al discorso sul bollettino *MS03-26*, l'effetto immediato è l'impatto di questa pubblicazione nei giorni seguenti, fu quello di creare scompiglio fra amministratori e utenti Windows, messi in ansia da un problema ancora poco chiaro e non manifestatosi – per fortuna –in nessun modo, visto che gli "hacker" del gruppo LSD, con grande spirito e coscienza, al momento della pubblicazione si rifiutarono di rivelare al mondo intero il loro exploit.
Ma Internet è la rete delle reti e spesso basta poco a scatenare un putiferio, specie quando si cercano informazioni non ancora esistenti e si hanno riferimenti ambigui. Il sistema RPC non è certo immune a bug e vulnerabilità e, nel corso del tempo, sono già stati sottoposti al pubblico diverse insicurezze, passate però stranamente inosservate.
Sarà stato quindi per caso o per errore che una serie di post apparsi in rete e di riferimenti al problema dell'RPC, abbiano portato in risalto un secondo bug del meccanismo RPC, documentato dal vecchio bollettino *MS03-10*, relativo questa volta a un problema di *Denial of Service*, scoperto dall'hacker Lion (*www.xfocus.org*) e dal gruppo ImmunitySec (*www. immunitysec.com*).
Non contento del massacro in corso verso il servizio RPC, si aggiunge infine alla lista dei "cattivi" una terza figura, *Benjurry* (*benjurry@xfocus.org*), che, per ironia della sorte, pubblica sempre nello stesso periodo, un altro exploit relativo ad una vulnerabilità di RPC e dell'interfaccia DCOM.
A questo punto la confusione in rete è totale: sia ben chiaro, le vulnerabilità del bollettino MS03-10 e quella pubblicata da Benjurry riguardano sempre l'RPC, ma sono ben diverse e non hanno nulla a che a vedere con quanto documentato nel bollettino *MS03-26*.

IL PUNTO DELLA SITUAZIONE

In sostanza si tratta di tre casi di bug

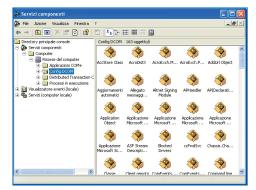



Fig. 3: **L'utility \WINDOWS\SYSTEM32\DCOMCFG.EXE permette di configurare DCOM e di verificare quali sono gli oggetti registrati all'interno di tale interfaccia. Per disabilitare l'interfaccia DCOM basta seguire la procedura riportata in queste pagine.**

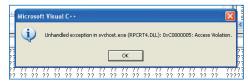



Fig. 4: **L'attacco di Denial of Service descritto nel bollettino MS03-010 sotto Windows XP causa un crash del servizio RPC (localizzato nella libreria RPCRT4.DLL) che manda in panico il ServiceHost del sistema operativo.**

Come disattivare l'interfaccia DCOM

All'interno del registro di configurazione di Windows 2000/XP è presente la chiave "EnableDCOM" alla voce HKLM\Software\Microsoft\OLE. Per disabilitare DCOM basta semplicemente cambiare il valore di questa chiave spostandolo da "Y" (yes) su "N" (no). Quando DCOM è disabilitato tutte le chiamate all'interfaccia vengono rifiutate (il caller riceverà un codice d'errore del tipo RPC_S_SERVER_UNVAILABLE).

Fig. 5: **Su Windows XP l'effetto immediato, che segue la notifica di arresto inaspettato del servizio RPC dovuto all'exploit, è quello di un reboot forzato da AUTHORITY \SYSTEM in persona!.**

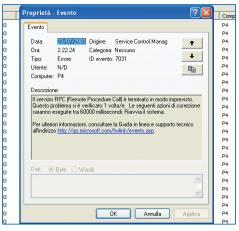



Fig. 6: Nel Visualizzatore Eventi del sistema operativo è possibile trovare ulteriori dettaglio sugli effetti dell'attacco DoS contro il servizio RPC.

Advisory	Bollettivo Microsoft	Tipo	Exploit
SPIKE www.securiteam.com/windowsntfocus/6G00B2K5PM.html	MS03-010 www.microsoft.com/technet/security/bulletin/MS03-010.asp	DoS	SI www.securiteam.com/exploits/6V00P0K5SE.html
LSD Team *lsd-pl.net/special.html*	MS03-026 www.microsoft.com/technet/security/bulletin/MS03-026.asp	Remote Execution	NO
benjurry@xfocus.org *www.securityfocus.com/archive/1/329755*	-	DoS, Privilege Escalation	SI www.securityfocus.com/archive/1/329755

Tab. 1: Bug relativi al servizio RPC.

diversi, ma tutti riguardanti RPC e la porta 135. La Tabella 1 riassume in modo chiaro la situazione dei bug e degli exploit disponibili in rete al momento.

Fig. 7: In Windows 2000 il reboot forzato del sistema fortunatamente non avviene, ma l'aggressione a RPC riesce lo stesso, causando l'arresto del servizio, come si può notare dagli errori riportati dal sistema operativo.

affligge soltanto Windows 2000, concentreremo la nostra attenzione sull'exploit relativo all'MS03-010.

Il codice C++ relativo a questo exploit (allegato per ragioni di spazio nel CD-ROM della rivista) è un potente *nuker* che interferisce via rete col servizio RPC attraverso la porta 135, mandando il servizio in crash (Fig. 4) attraverso una richiesta malformata.

Sotto Windows XP è necessario ripetere l'attacco almeno 2 volte per avere qualche effetto, riuscendo a provocare addirittura il reboot forzato (Fig. 5) del sistema.

Sotto Windows 2000 invece i danni sono contenuti e l'exploit si limita a killare il servizio RPC (Fig. 7) senza però mandare in crash l'intero sistema operativo: la chiusura di RPC (Fig. 8) provoca comunque qualche problema che interessa le risorse di rete e i processi legati a RPC.

Elia Florio

Fig. 8: La terminazione di RPC rende comunque il sistema poco stabile e alcune funzionalità (risorse di rete, messenger, finestre di proprietà) potrebbero risultare compromesse. Un reboot, in questi casi, risolve tutti i problemi e fa ripartire il servizio RPC.

CRASHARE WINDOWS ATTRAVERSO RPC

Poiché sull'MS03-026 non esiste ancora un exploit pubblico (chissà quando verrà rilasciato dagli LSD? avremo una nuova ondata di worm?) e poiché il bug scoperto da Benjurry

Ultim'ora

Dopo lunghe attese il codice sorgente per l'exploit è stato rilasciato al pubblico ma non dagli scopritori (il team LSD), bensì dal team concorrente Xfocus.org."

Sul Web

Bollettini Microsoft MS03-010 e MS03-026
http://www.microsoft.com/technet/security/bulletin/MS03-010.asp
http://www.microsoft.com/technet/security/bulletin/MS03-026.asp

Advisory rilasciato dal CERT relativo al problema scoperto dal team LSD
http://www.cert.org/advisories/CA-2003-16.html

Siti con exploit e vulnerabilità
http://www.k-otik.com
http://www.cnhonker.net/Down/index.php

RPC Nuker, con interfaccia grafica
http://uncelhacker.net/Downloads/5/nuker/rpcnuke9x-nt-xp.zip

Tips&Tricks

I trucchi del mestiere

La rubrica raccoglie trucchi e piccoli pezzi di codice che solitamente non trovano posto nei manuali, ma sono frutto dell'esperienza di chi programma. Alcuni trucchi sono proposti dalla Redazione, altri provengono da una ricerca sulla Rete delle Reti, altri ancora ci giungono dai lettori. Chi vuole contribuire potrà inviarci i suoi tips&tricks preferiti che, una volta scelti, verranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory \tips\ o sul Web: cdrom.ioprogrammo.net.

▶ Visual Basic

Numeri binari con Visual Basic

Una serie di funzioni che permettono di "lavorare" con i numeri binari; nella fattispecie sono implementate le seguenti procedure e/o funzioni:

- *DecTOBin* e *BinTODec* permettono di convertire valori di byte nel loro corrispettivo binario e vice versa.
- *GetLowHightValueBits* restituisce i valori dei bit alti e bassi all'interno di un byte.
- *SplitLongValues, SplitIntegerValues, MergeLongValues* e *MergeIntegerValues* scompogno e ricompongono numeri *Integer* e *Long* nei loro sotto byte.
- *ShiftByte* permette di compiere operazioni tipo *Mid$* sui valori dei Bit di un singolo byte restituendo poi il valore risultante.

Tip fornito dal Sig. S.Tubini

```
Option Explicit

Private Declare Sub CopyMemory Lib "kernel32" Alias
                    "RtlMoveMemory" (Destination As Any, Source As Any,
                                                    ByVal Length As Long)
Public Enum ShiftType
    RightShift = 1
    LeftShift = 2
End Enum
Public Sub GetLowHightValueBits(bValue As Byte, lValue As Byte,
                                                        hValue As Byte)
Dim BitArr(7) As Byte, LBArr(7) As Byte, HBArr(7) As Byte
    DecTOBin bValue, BitArr()
    LBArr(0) = BitArr(7)
    LBArr(1) = BitArr(6)
    LBArr(2) = BitArr(5)
    LBArr(3) = BitArr(4)
    HBArr(0) = BitArr(3)
    HBArr(1) = BitArr(2)
    HBArr(2) = BitArr(1)
    HBArr(3) = BitArr(0)
    lValue = BinTODec(LBArr())
    hValue = BinTODec(HBArr())
End Sub
Public Sub DecTOBin(bValue As Byte, BitArr() As Byte)
Dim I As Long
    For I = 1 To bValue Step 1
      If BitArr(7) = 1 Then
        BitArr(7) = 0
        If BitArr(6) = 1 Then
          BitArr(6) = 0
          If BitArr(5) = 1 Then
            BitArr(5) = 0
            If BitArr(4) = 1 Then
              BitArr(4) = 0
              If BitArr(3) = 1 Then
                BitArr(3) = 0
                If BitArr(2) = 1 Then
                  BitArr(2) = 0
                  If BitArr(1) = 1 Then
                    BitArr(1) = 0
                    BitArr(0) = 1
                  Else
                    BitArr(1) = 1
                  End If
                Else
                  BitArr(2) = 1
                End If
              Else
                BitArr(3) = 1
              End If
            Else
              BitArr(4) = 1
            End If
          Else
            BitArr(5) = 1
          End If
        Else
          BitArr(6) = 1
        End If
      Else
        BitArr(7) = 1
      End If
    Next
End Sub
Public Function BinTODec(BitIn() As Byte) As Byte
Dim I As Long, BitArr(7) As Byte
    For I = 1 To 255 Step 1
      If BitArr(7) = 1 Then
        BitArr(7) = 0
        If BitArr(6) = 1 Then
          BitArr(6) = 0
          If BitArr(5) = 1 Then
            BitArr(5) = 0
            If BitArr(4) = 1 Then
              BitArr(4) = 0
              If BitArr(3) = 1 Then
```

```
                        BitArr(3) = 0
                        If BitArr(2) = 1 Then
                            BitArr(2) = 0
                            If BitArr(1) = 1 Then
                                BitArr(1) = 0
                                BitArr(0) = 1
                            Else
                                BitArr(1) = 1
                            End If
                        Else
                            BitArr(2) = 1
                        End If
                    Else
                        BitArr(3) = 1
                    End If
                Else
                    BitArr(4) = 1
                End If
            Else
                BitArr(5) = 1
            End If
        Else
            BitArr(6) = 1
        End If
    Else
        BitArr(7) = 1
    End If
    If BitIn(0) = BitArr(0) And BitIn(1) = BitArr(1) And BitIn(2)
        R= BitArr(2) And BitIn(3) = BitArr(3) And BitIn(4) = BitArr(4)
        And BitIn(5) = BitArr(5) And BitIn(6) = BitArr(6) And BitIn(7) =
                                                          BitArr(7) Then
        BinTODec = I
        Exit For
    End If
  Next
End Function
Public Sub SplitLongValues(lValue As Long, ByteArr() As Byte)
    CopyMemory ByteArr(0), lValue, 4
End Sub
Public Sub SplitIntegerValues(iValue As Integer, ByteArr() As Byte)
    CopyMemory ByteArr(0), lValue, 2
End Sub
Public Function MergeLongValues(ByteArr() As Byte) As Long
    CopyMemory MergeLongValues, ByteArr(), 4
End Function
Public Function MergeIntegerValues(ByteArr() As Byte) As Integer
    CopyMemory MergeIntegerValues, ByteArr(), 2
End Function
Public Function ShiftByte(bValue As Byte, nBitStart As Byte,
                                        nSize As Byte, ShiftMode As
ShiftType) As Byte
On Local Error Resume Next
Dim bArr(7) As Byte
Dim NewbArr(7) As Byte
Dim I As Long,, nStep As Long, nSkip As Long
If (nBitStart + nSize * nStep) < 8 Or (nBitStart + nSize * nStep) >
                                              -1 And nBitStart < 8 Then
    If ShiftMode = LeftShift Then
        nStep = -1
    ElseIf ShiftMode = RightShift Then
        nStep = 1
    End If
    DecTOBin bValue, bArr()
    nSkip = 7
    For  I = (7 - nBitStart) To (7 - (nBitStart + nSize - 1)) Step nStep
        NewbArr(nSkip) = bArr(I)
        nSkip = nSkip - 1
    Next
End If
If Err <> 0 Then Err = 0: Exit Function
ShiftByte = BinTODec(NewbArr())
End Function
```

Grafici a torta? Semplice...

Un piccolo ma funzionale tip che consente di creare grafici a torta senza utilizzare nessun ActiveX di supporto.

Tip fornito dal Sig. Luciano Busetti

Il codice del progetto è presente nella sezione codice del Cd-Rom allegato o sul Web: *cdrom.ioprogrammo.net*

Alla ricerca del file perduto

Un tip che consente di cercare un qualunque file contenuto in qualunque hard-disk installato nel personal computer. L'intera operazione di ricerca è affidata a delle API di sistema in grado di interagire direttamente con il FileSystem di Windows. Per la ricerca è anche possibile inserire il carattere jolly *"*"Es.: "*.txt";* in output fornisce:

1. Il numero di file trovati
2. Il numero di cartelle nelle quali ha cercato
3. La grandezza complessiva in Byte dei file trovati
4. Come parametro di ritorno una stringa contenente i percorsi dei file che sono stati trovati separati dal carattere "|"

Tip fornito dal Sig. Alessandro Castaldo

```
'Creare un Form con:
'un TextBox nominato Text1 per la Cartella di ricerca
'un TextBox nominato Text2 per il nome del file da cercare
'un CommandButton nominato Command1 per avviare la ricerca
Option Explicit
Private Declare Function FindFirstFile Lib "kernel32" Alias "FindFirstFileA"
 (ByVal lpFileName As String, lpFindFileData As WIN32_FIND_DATA) As Long
 Private Declare Function FindNextFile Lib "kernel32" Alias "FindNextFileA"
 (ByVal hFindFile As Long, lpFindFileData As WIN32_FIND_DATA) As Long
            Private Declare Function GetFileAttributes Lib "kernel32" Alias
                "GetFileAttributesA" (ByVal lpFileName As String) As Long
Private Declare Function FindClose Lib "kernel32" (ByVal hFindFile As Long)
                                                                  As Long
Const MAX_PATH = 260
Const MAXDWORD = &HFFFF
Const INVALID_HANDLE_VALUE = -1
Const FILE_ATTRIBUTE_ARCHIVE = &H20
Const FILE_ATTRIBUTE_DIRECTORY = &H10
Const FILE_ATTRIBUTE_HIDDEN = &H2
Const FILE_ATTRIBUTE_NORMAL = &H80
Const FILE_ATTRIBUTE_READONLY = &H1
Const FILE_ATTRIBUTE_SYSTEM = &H4
```

```
Const FILE_ATTRIBUTE_TEMPORARY = &H100
Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type
Private Type WIN32_FIND_DATA
    dwFileAttributes As Long
    ftCreationTime As FILETIME
    ftLastAccessTime As FILETIME
    ftLastWriteTime As FILETIME
    nFileSizeHigh As Long
    nFileSizeLow As Long
    dwReserved0 As Long
    dwReserved1 As Long
    cFileName As String * MAX_PATH
    cAlternate As String * 14
End Type
Private Function StripNulls(OriginalStr As String) As String
    If (InStr(OriginalStr, Chr(0)) > 0) Then
        OriginalStr = Left(OriginalStr, InStr(OriginalStr, Chr(0)) - 1)
    End If
    StripNulls = OriginalStr
End Function
Private Function FindFilesAPI(ByVal Path As String, ByVal SearchStr As
        String, ByRef FileCount As Long, ByRef DirCount As Long, ByRef
                                                FoundFiles As String)
    Dim FileName As String
    Dim DirName As String
    Dim dirNames() As String
    Dim nDir As Long
    Dim i As Long
    Dim hSearch As Long
    Dim WFD As WIN32_FIND_DATA
    Dim Cont As Long
    DoEvents
    If Right(Path, 1) <> "\" Then Path = Path & "\"
    nDir = 0
    ReDim dirNames(nDir)
    Cont = True
    hSearch = FindFirstFile(Path & "*", WFD)
    If hSearch <> INVALID_HANDLE_VALUE Then
        Do While Cont
        DirName = StripNulls(WFD.cFileName)
        If (DirName <> ".") And (DirName <> "..") Then
            If GetFileAttributes(Path & DirName) And
                                        FILE_ATTRIBUTE_DIRECTORY Then
                dirNames(nDir) = DirName
                DirCount = DirCount + 1
                nDir = nDir + 1
                ReDim Preserve dirNames(nDir)
            End If
        End If
        Cont = FindNextFile(hSearch, WFD)
        Loop
        Cont = FindClose(hSearch)
    End If
    hSearch = FindFirstFile(Path & SearchStr, WFD)
    Cont = True
    If hSearch <> INVALID_HANDLE_VALUE Then
        While Cont
            FileName = StripNulls(WFD.cFileName)
            If (FileName <> ".") And (FileName <> "..") Then
                FindFilesAPI = FindFilesAPI + (WFD.nFileSizeHigh *
                                        MAXDWORD) + WFD.nFileSizeLow
                FileCount = FileCount + 1
                If FoundFiles <> "" Then FoundFiles = FoundFiles & "|"
                FoundFiles = FoundFiles & Path & FileName
            End If
            Cont = FindNextFile(hSearch, WFD)
        Wend
        Cont = FindClose(hSearch)
    End If
    If nDir > 0 Then
        For i = 0 To nDir - 1
            FindFilesAPI = FindFilesAPI + FindFilesAPI(Path & dirNames(i) &
                    "\", SearchStr, FileCount, DirCount, FoundFiles)
        Next i
    End If
End Function
Public Function FindFile(ByVal SearchPath As String, ByVal SearchStr As
String, ByRef FileCount As Long, ByRef DirCount As Long,
                                    ByRef FileSize As Long) As String
    FileSize = FindFilesAPI(SearchPath, SearchStr, FileCount, DirCount,
                                                            FindFile)
End Function
Sub Command1_Click()
    Dim FileSize As Long
    Dim FileCount As Long
    Dim DirCount As Long
    Dim OutMessage As String
    Dim FoundFiles As String
    Screen.MousePointer = vbHourglass
    Command1.Enabled = False
    FoundFiles = FindFile(Text1.Text, Text2.Text, FileCount, DirCount,
                                                            FileSize)
    Command1.Enabled = True
    Screen.MousePointer = vbDefault
    FoundFiles = Replace(FoundFiles, "|", vbCrLf)
    OutMessage = OutMessage & FoundFiles & vbCrLf
    OutMessage = OutMessage & vbCrLf
    OutMessage = OutMessage & FileCount & " file trovati in "
                                        & DirCount + 1 & " cartelle"
    OutMessage = OutMessage & vbCrLf
    OutMessage = OutMessage & "La dimensione dei file trovati in " &
    Text1.Text & " è " & Format(FileSize, "#,###,###,##0") & " Byte"
    MsgBox OutMessage, , "Risultato ricerca"
End Sub
```

C++/C#

Una classe in C++ per manipolare le date

Una classe Data che può risultare comoda per chi fa programmi che utilizzano questo tipo di informazione.

Tip fornito dal Sig. Eugenio Dei Giudici

```
// FILE Data.h - Header File
#ifndef DATAH
#define DATAH
#include <iostream>
#include <string>
using namespace std;
class Data {
// Operatore d'uscita (output)
friend ostream& operator<<(ostream&, const Data&);
// Operatore d'ingresso (input)
friend istream& operator>>(istream&, Data&);
// Operatori di confronto
friend bool operator==(const Data&, const Data&);
friend bool operator!=(const Data&, const Data&);
friend bool operator<(const Data&, const Data&);
friend bool operator<=(const Data&, const Data&);
friend bool operator>(const Data&, const Data&);
friend bool operator>=(const Data&, const Data&);
public:
   Data(int,int,int);           // Costruttore
   Data(const Data&);           // Costruttore di copia
   Data();                      // Costruttore di default
   Data& operator=(const Data&); // Operatore d'assegnamento
   int getAnno() const { return anno; }  // Funzioni che restituiscono
   int getMese() const { return mese; } // i valori delle variabili
   int getGiorno() const { return giorno; }        // di stato
   void modG(int);              // Funzioni atte
   void modM(int);  // a modificare
   void modA(int);  // le variabili di stato
private:
   int anno;                    // Variabili
   int mese;                    // di
   int giorno;                  // stato
// Funzione di verifica dell'effettiva esattezza di una possibile Data
   bool controlla (int,int,int);
}; // Data
#endif
// File Data.cpp - Implementazione
#include "Data.h"
#include <time.h>
Data::Data(int g,int m,int a) {
   if(!controlla(g,m,a)) throw new exception();
      giorno=g;
      mese=m;
      anno=a;
} // costruttore
Data::Data(const Data& d) {
   giorno=d.giorno;
   mese=d.mese;
   anno=d.anno;
} // costruttore di copia
Data::Data() {
   int g;
   int m;
   int a;
   time_t t=time(NULL);
   char* s=ctime(&t); /* stringa contenente la data corrente sottoforma di:
                  giorno_della_settimana mese giorno ora anno*/
   char* p;  /* puntatore a carattere necessario a puntare il primo
                                          carattere utile dei token */
   p=strtok(s," ");   // prendo il primo token che non mi serve
   p=strtok(NULL," ");  /* prendo il token successivo sostituendo NULL
   \al carattere 0 del token precedente, modificando così la stringa di
                                                           partenza */
   if(strcmp(p,"Jan")==0) m=1;
   if(strcmp(p,"Feb")==0) m=2;
   if(strcmp(p,"Mar")==0) m=3;
   if(strcmp(p,"Apr")==0) m=4;
   if(strcmp(p,"May")==0) m=5;
   if(strcmp(p,"Jun")==0) m=6;
   if(strcmp(p,"Jul")==0) m=7;
   if(strcmp(p,"Aug")==0) m=8;
   if(strcmp(p,"Sep")==0) m=9;
   if(strcmp(p,"Oct")==0) m=10;
   if(strcmp(p,"Nov")==0) m=11;
   if(strcmp(p,"Dec")==0) m=12;
   p=strtok(NULL," ");          // prendo il token col giorno
   g=atoi(p);          // converto il giorno da string in int
   p=strtok(NULL," ");  // prendo il token con l'ora e lo scarto subito
   p=strtok(NULL," ");          // prendo il token con l'anno
   a=atoi(p);                   // converto l'anno da string in int
   if(!controlla(g,m,a)) throw new exception();
   giorno=g;
   mese=m;
   anno=a;
} // costruttore di default (ritorna la data corrente dell'orologio di
                                                            sistema)
Data& Data::operator=(const Data& d) {
          giorno=d.giorno;
          mese=d.mese;
          anno=d.anno;
          return *this;
} // operator=
void Data::modG(int g) {
          if (!controlla(g, mese, anno)) throw new exception();
          giorno=g;
} // modG
void Data::modM(int m) {
          if (!controlla(giorno, m, anno)) throw new exception();
          mese=m;
} // modM
void Data::modA(int a) {
          if (!controlla(giorno, mese, a)) throw new exception();
          anno=a;
} // modA
bool Data::controlla (int g,int m,int a) {
   if (g<1 || g>31) return false;
   if (m<1 || m>12) return false;
   if (a<=-1000000000 || a>=1000000000) return false;
                        // limiti totalmente arbitrari (cifra tonda!)
   if (g==31 && (m==4 || m==6 || m==9 || m==11) ) return false;
   if (m==2)
   if (g>29 || g==29 && a%4!=0 ) return false;
   return true;
} // controlla
// Overloading degli operatori
// Operatore d'uscita (output)
ostream& operator<<(ostream& os, const Data& d) {
```

```
os<<d.giorno<<"/"<<d.mese<<"/"<<d.anno;
return os;
} // operator<<
// Operatore d'ingresso (input)
istream& operator>>(istream& is, Data& d) {
        int g,m,a;
        char* c=new char[50];
        cout<<"Inserisci il giorno: ";
        is.getline(c,50);
        g=atoi(c);
        cout<<"Inserisci il mese: ";
        is.getline(c,50);
        m=atoi(c);
        cout<<"Inserisci l'anno: ";
        is.getline(c,50);
        a=atoi(c);
        delete [] c;
        if(!d.controlla(g,m,a)) throw new exception();
        d.modG(g);
        d.modM(m);
        d.modA(a);
        return is;
} // operator>>
// Operatori di confronto
bool operator==(const Data& d1, const Data& d2) {
  return (d1.giorno==d2.giorno && d1.mese==d2.mese &&
                                    d1.anno==d2.anno);
} // operator==
bool operator!=(const Data& d1, const Data& d2) {
  return (d1.giorno!=d2.giorno || d1.mese!=d2.mese || d1.anno!=d2.anno);
} // operator!=
bool operator<(const Data& d1, const Data& d2) {
```

IL TIP-ONE del mese

Nascondere il menu contestuale dei filmati Macromedia Flash di una Windows Forms

Il tip, in linguaggio C#, spiega come inserire un filmato Macromedia Flash in una applicazione Windows .NET e nasconderne completamente il menu contestuale.
Tip fornito dal Sig. Sergio Turolla

Come prima operazione, utilizzando l'applicazione "aximp" fornita a corredo con l'SDK del Framework .NET, si importa l'ActiveX di Flash utilizzando la seguente sintassi:

aximp "c:\WINDOWS\system32\Macromed\Flash\Flash.ocx" /source

(NB: Si presuppone che il player Flash 6 sia correttametne installato nel suo path di default (*"c:\WINDOWS\system32\ Macromed\Flash\Flash.ocx"*, se non è cosi lo si puo scaricare dal sito della Macromedia *"www.macromedia.com"*). Saranno generati tre file di cui solo due necessari ai nostri scopi: *"ShockwaveFlashObjects.dll"* e soprattutto *"AxShockwaveFlashObjects.cs"* (il sorgente del wrapper .NET per l'activeX di flash). Successivamente, editando il file *"AxShockwaveFlashObjects.cs"* generato automaticamente dall'utility *"aximp"* si deve aggiungere il seguente codice alla classe *AxShockwaveFlash:*

```
public class AxShockwaveFlash : System.Windows.Forms.AxHost {
  // Costante dell'evento da intercettare
  private readonly int rightMouseClickMessage = 516;
  // Disabilita il menu contestuale
  protected override void WndProc(ref System.Windows.Forms.Message m)
  {
    // Intercetta ed esegue tutti i messaggi tranne
    // quello del tasto destro del mouse che causa la
    // visualizzazione del menu contestuale
    if (m.Msg != rightMouseClickMessage)
       base.WndProc(ref m); }
   ...
...
```

Questo codice fa si che vengano intercettati e gestiti, dal componente Flash, tutti i messaggi ad eccezione di quello generato dalla pressione del tasto destro del mouse. In questo modo il player Flash, non ricevendo mai il messaggio di Windows, che lo informa della pressione del tasto destro del mouse, inibisce la visualizzazione del menu contestuale associato all'evento. E' ora possibile utilizzare la classe all'interno dei nostri applicativi Windows Forms, ad esempio in questo modo:

```
using System;
using System.Drawing;
using System.Windows.Forms;
using AxShockwaveFlashObjects;
namespace FlashTest
{
  public class FlashForm  : System.Windows.Forms.Form
  {
    private AxShockwaveFlash flashControl;
    public FlashForm ()
    {
        this.Text = "FlashTest";
        this.Size = new Size(500,400);
        this.Load += new EventHandler(this.myWindow_Load);
        // crea il controllo flash e lo aggiung al form
        this.flashControl = new AxShockwaveFlash();
        this.flashControl.Location = new Point(10,10);
        this.flashControl.Size = new Size(300,300);
        this.Controls.Add(this.flashControl); }
      public void myWindow_Load(object sender,EventArgs e)
      {
        // Carica il filmato flash
        // Per un corretto funzionamento caricare il filmato
        // sempre sull'evento Load del form o successivamente
        // e mai nel costruttore
        this.flashControl.Movie = System.IO.Path.Combine(
                     Application.StartupPath,"Login.swf"); }
        public static void Main()
        { Application.Run(new FlashForm());}
  }
}
```

In ultimo, per compilare l'applicativo, bisogna ricordarsi di aggiungere un riferimento al file *"ShockwaveFlashObjects.dll"* generato da *"aximp"*, ad esempio eseguire:

csc /target:winexe FlashForm.cs AxShockwaveFlashObjects.cs/r:ShockwaveFlashObjects.dll

```
  return (d1.anno<d2.anno || d1.anno==d2.anno && d1.mese<d2.mese ||
   d1.anno==d2.anno && d1.mese==d2.mese && d1.giorno<d2.giorno);
} // operator<
bool operator<=(const Data& d1, const Data& d2) {
return (d1.anno<d2.anno || d1.anno==d2.anno && d1.mese<d2.mese ||
d1.anno==d2.anno && d1.mese==d2.mese && d1.giorno<d2.giorno ||
 d1.giorno==d2.giorno && d1.mese==d2.mese && d1.anno==d2.anno);
} // operator<=
bool operator>(const Data& d1, const Data& d2) {
  return (d1.anno>d2.anno || d1.anno==d2.anno && d1.mese>d2.mese ||
                    d1.anno==d2.anno && d1.mese==d2.mese &&
                                       d1.giorno>d2.giorno);
} // operator>
bool operator>=(const Data& d1, const Data& d2) {
  return (d1.anno>d2.anno || d1.anno==d2.anno && d1.mese>d2.mese ||
                    d1.anno==d2.anno && d1.mese==d2.mese &&
                   d1.giorno>d2.giorno ||d1.giorno==d2.giorno &&
        d1.mese==d2.mese && d1.anno==d2.anno); } // operator>=
```

ASP

Creare delle immagini formato thumbnail

In un sito Web è sempre preferibile utilizzare delle immagini di formato ridotto che, al semplice click del mouse, possano mostrarsi nella loro interezza. Il tip mostra come realizzare delle immagini di formato ridotto, note come *thumbnail*.

```
<%
Const MINPIXELS = 100
Set objImageSize = Server.CreateObject("ImgSize.Check")
objImageSize.FileName = Server.MapPath("IMG.jpg")
ImageHeight = objImageSize.Height
ImageWidth = objImageSize.Width
If ImageHeight > ImageWidth Then
   NewHeight = Cint(ImageHeight*MINPIXELS/ImageWidth)
   NewWidth = MINPIXELS
Else
   NewWidth = Cint(ImageWidth*MINPIXELS/ImageHeight)
   NewHeight = MINPIXELS
End If
Set objImageSize = Nothing
'*** Creazione dell'immagine thumbnail
Set Image = Server.CreateObject("AspImage.Image")
Image.LoadImage(Server.MapPath("IMG.jpg"))
Image.FileName = Server.MapPath("IMG_piccola.jpg")
Image.ImageFormat = 1
Image.JPEGQuality = 70
Image.Resize NewWidth,NewHeight
Image.SaveImage
Set Image = Nothing %>
```

Creare un documento Word da ASP

Ecco come, in modo semplice e veloce, è possibile creare un documento Word da una comune pagina ASP.
La pagina richiede l'immissione dei semplici dati e da questi ne propone una visualizzazione in Microsoft Word.

```
<form action="word_create.asp">
Name: <input type="text" name="Nome" size="50" maxlength="100">
Email: <input type="text" name="Email" size="50" maxlength="100">
Commenti:
<textarea cols="50" rows="10" name="commenti"></textarea>
<input type="submit" value="Submit">
</form>
```

Di seguito come creare, di fatto, il documento Microsoft Word:

```
<% Response.ContentType = "application/msword" %>
<html>
<% strNome = Request.Querystring("Nome")
strEmail = Request.Querystring("Email")
strCommenti = Request.Querystring("Commenti") %>
<head>
</head>
<body>
<p align="right"><%=formatdatetime(now,2)%></p>
Dear <%= strnome %>:
Il mio indirizzo email è: <%= stremail %>
Ecco qui i commenti: <%= strCommenti %>
</body>
</html>
```

Robotica: hardware e software (parte seconda)

Mano meccanica: i sensori del tatto

Elettronica

L'uomo è in grado di manipolare una infinita quantità di oggetti ed utensili, grazie ad un importantissimo senso: il tatto. Anche la nostra mano meccanica ha bisogno di questa caratteristica, per poter maneggiare efficacemente svariati oggetti. Una serie di sei sensori di pressione ed un sensore di prossimità a raggi infrarossi conferiranno al nostro braccio la capacità di capire se ha posizionato la mano in modo corretto e se ha afferrato con efficacia l'oggetto da manipolare.

L'importanza di quelle capacità, riconosciute comunemente come i 'cinque sensi', appare evidente a chiunque. Quanto il tatto possa essere fondamentale nelle manipolazione degli oggetti, può apparire scontato, anche se i meccanismi che lo regolano sono tutt'altro che semplici e vengono gestiti dalla parte più antica e profonda del nostro cervello. Per maggiori informazioni sul braccio meccanico, sulle interfacce relative e sull'apparecchiatura 'PC Explorer light' è possibile visitare il sito: *http://web.tiscali.it/spuntosoft/*. Alcune reazioni correlate al tatto sono infatti pressochè automatiche e quantomai rapide, perché provengono dal bagaglio di conoscenze registrate nella nostra memoria biologica in seguito a milioni di anni di evoluzione. Pensiamo infatti a quando tocchiamo un oggetto troppo caldo: istintivamente ritraiamo la mano per evitare di scottarci. Altri meccanismi prevedono invece la conoscenza e l'acquisizione delle caratteristiche di un oggetto da manipolare, quali il suo peso, le sue dimensioni ed il suo stato fisico, nonché la sua forma. Proviamo ad afferrare un oggetto ad occhi chiusi: come facciamo a capire se siamo in grado di afferrarlo correttamente e di sollevarlo? Come riusciamo a comprendere se la nostra mano è adeguata alla manipolazione dell'oggetto, oppure se questo è troppo piccolo o troppo grande, oppure troppo pesante?

Fig. 1: La mano meccanica è stata progettata in modo tale che possa maneggiare oggetti di peso e dimensioni ragguardevoli: nella figura si nota il meccanismo in azione con una comune tazza da caffè.

Fig. 2: Al termine della lettura di queste pagine, il lettore sarà in grado di realizzare il controllo di una mano meccanica completa di sensori del tatto.

La risoluzione di questo problema è possibile, anche ad occhi chiusi, perché su tutta la superficie della nostra pelle sono presenti una miriade di sensori di pressione e temperatura, tutti collegati, attraverso il sistema nervoso al nostro cervello. In questo modo siamo in grado di capire se la nostra mano ha toccato l'oggetto ed in quale punto preciso, spostando e muovendo le dita possiamo capire quanto è grande ed afferrandolo e cercando di sollevarlo possiamo stabilire se

Quesiti all'autore

L'autore è lieto di rispondere ai quesiti dei lettori sull'interfacciamento dei PC all'indirizzo:
luca.spuntoni@ioprogrammo.it

Elettronica

Mano
meccanica:
i sensori del tatto

la presa è adeguata, oppure se l'oggetto sta scivolando perché è troppo pesante o perché è stato afferrato in modo non appropriato. In questa sede vogliamo conferire alla mano meccanica, presentata nell'appuntamento precedente, la capacità sensoriale idonea alla manipolazione di oggetti di dimensioni e peso anche consistenti. Il braccio meccanico verrà dotato di sei sensori di pressione, ed uno di prossimità, a raggi infrarossi, per il corretto posizionamento della mano e per l'appropriata manipolazione degli oggetti, sia in termini spaziali che di pressione da esercitare.

I SENSORI DELLA MANO MECCANICA

Per potere conferire alla nostra mano meccanica la stessa sensibilità di un arto umano, avremmo bisogno di una quantità enorme di sensori di pressione, umidità e temperatura: l'implementazione del congegno sarebbe troppo complesso e la realizzazione risulterebbe troppo pesante sia in termini di peso che economici.

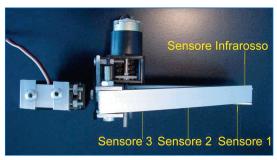



Fig. 3: La mano meccanica è dotata di sei sensori di pressione ed uno di prossimità a raggi infrarossi.

I componenti necessari

- N6 Microinterruttori;
- N1 LED Infrarosso;
- N1 Fototransistor;
- N4 Resistenze 4,7 K _ W;
- N1 Resistenze 1K _ W.

Il software e l'hardware di gestione sarebbero inoltre molto complicati: dobbiamo quindi semplificare la struttura sensoriale, in modo tale che risulti pratica e realizzabile, nonché proporzionata allo scopo che ci siamo prefissi, ossia la manipolazione di oggetti, anche di peso e dimensioni considerevoli. La dotazione di sensori della nostra mano sarà notevolmente migliore di quella disponibile allo stato attuale per quasi tutti i bracci meccanici rivolti agli hobbisti: molte apparecchiature industriali, inoltre, invidieranno le capacità sensoriali della nostra mano. Come sensori di pressione utilizziamo sei microinterruttori in miniatura, montati su due supporti basculanti, alloggiati su ciascuna delle dita, in modo tale da avere due coppie di sensori all'estremità *(Sensore 1)*, a metà dito *(Sensore 2)* ed alla fine *(Sensore 3)*. Questa disposizione permette di individuare il punto nel quale si sta prendendo l'oggetto, oltre al fatto che è possibile stabilire se si sta verificando uno scivolamento, rilevato da una diversa pressione sui sensori. Qualora il posizionamento non sia soddisfacente, è possibile riaprire la mano, riposizionarla nel punto più opportuno e riprovare la presa, fino al raggiungimento del risultato voluto. E' possibile inoltre misurare la pressione desiderata sull'oggetto, come risultato della commutazione di uno, due o tutti i sensori: è possibile sostituire i microinterruttori con sensori di pressione analogici, qualora si necessiti una maggiore precisione, questa soluzione comporta, però, un notevole appesantimento del sistema, dal punto di vista fisico, della complessità tecnica e dal lato economico. Il sensore infrarosso è costituito da un trasmettitore e da un ricevitore IR, situati all'estremità della mano che permettono di rilevare la presenza di un oggetto all'interno della pinza. Questa caratteristica aggiuntiva, ovviamente non presente nella mano umana, facilita non di poco il corretto posizionamento della pinza meccanica: con questo sensore è possibile inoltre stabilire in qualche misura la dimensione dell'oggetto, per determinare se sia più conveniente afferrarlo con la punta delle dita o con l'intera mano. Ovviamente, per ottenere una capacità di manipolazione ottimale, è conveniente aggiungere una telecamera digitale al sistema ed una opportuna analisi dell'immagine, problematiche che affronteremo nel prosieguo dello sviluppo del nostro braccio meccanico. Completata la descrizione funzionale dei sensori, procediamo ad analizzarne le metodologie di implementazione, prima nella struttura hardware e poi nel programma di gestione software.

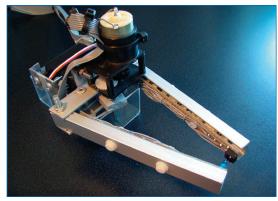



Fig. 4: Nell'immagine di figura è possibile notare un particolare dei sensori di pressione e di prossimità a raggi infrarossi: i sei sensori di pressione permettono di verificare in quale zona della mano meccanica è avvenuta la presa dell'oggetto.

L'IMPLEMENTAZIONE DEL CONTROLLO DEI SENSORI

La comprensione dell'implementazione hardware del controllo dei sensori della nostra mano meccanica, potrà essere di molto facilitata se nel prosieguo della sua analisi si seguirà la tabella riportata appresso, che riassume tutte le caratteristiche salienti del sistema, che verranno poi tradotte sotto forma di schema elettrico più avanti in queste pagine. I primi due segnali vengono riportati per completezza nella tabella e riguardano le linee di controllo del motore dell'apertura e chiusura della mano: per maggiori dettagli fare riferimento all'articolo relativo del numero precedente. Il bit *D2* corrisponde al diodo LED all'infrarosso, che funge da emettitore del sensore relativo al controllo della presenza di un oggetto all'interno della pinza.

PIN PC Master	SEGNALE	DIREZIONE	TIPO PORTA	PIN PORTA	DESCRIZIONE
2	D0	OUT	PARALLELA DATA D0	2	Open Hand
3	D1	OUT	PARALLELA DATA D1	3	Close Hand
4	D2	OUT	PARALLELA	4	INFRA RED sensor EMITTER
10	$\overline{\text{ACK}}$	IN STATUS S6	PARALLELA	10	INFRA RED sensor RECEIVER
11	BUSY	IN	PARALLELA STATUS /S7	11	Reserved
12	PE	IN	PARALLELA STATUS S5	12	Pression sensor 1
17	SLCT IN	IN	PARALLELA STATUS S4	17	Pression sensor 2
14	$\overline{\text{AUTO FD}}$	IN	PARALLELA STATUS S3	14	Pression sensor 3
18	GND	PARALLEL GND	PARALLELA	18-25	Signal Ground

Tab. 1: **Funzionalità dei diversi PIN PC MASTER.**

Elettronica

Mano
meccanica: i sensori del tatto

Per rilevare la presenza del fascio luminoso IR, viene utilizzata la linea corrispondente al bit *S6 (Porta di status, bit 6)*, che corrisponde al segnale di ingresso /ACK nello standard Centronics.

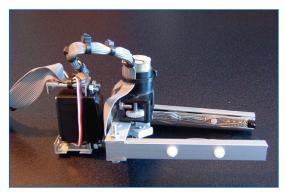



Fig. 5: **Da una analisi dei dettagli costruttivi della mano meccanica, si nota che è completamente costruita in alluminio, sono visibili l'attuatore della presa utilizzato in questa sede ed il servomeccanismo deputato alla rotazione del polso che analizzeremo nei prossimi appuntamenti.**

Le linee *S3, S4, S5,* che corrispondono rispettivamente a */AUTO FD, SLCT IN* e *PE,* vengono utilizzate per 'leggere' lo stato logico delle tre coppie di sensori di pressione: praticamente, se una di queste linee si trova allo stato logico *'High'* significa che uno dei due sensori posti su entrambe le dita è premuto, denotando il contatto con un oggetto od il fine corsa della mano, con conseguente contatto tra le due dita. Il lettore attento avrà notato che è stato riservato l'utilizzo del bit *S7:* su questa linea verranno letti serialmente altri 16 sensori del Robot. Il motivo di questa scelta è da imputarsi, oltre che a motivi di espandibilità del sistema, anche a ragioni relative alle diverse velocità di lettura dei dati paralleli citati finora e le tecniche hardware e software necessarie alla gestione seriale di 16 sensori su una sola linea. Per ora diciamo soltanto che si è preferito fornire una capacità 'preferenziale' alla mano meccanica, dal punto di vista della velocità di esecuzione, e per potere automatizzare la gestione della mano con l'utilizzo di un circuito di espansione dedicato opzionale che verrà presentato in futuro.

LO SCHEMA ELETTRICO

Lo schema elettrico traduce, sotto forma di circuito elettronico, quanto detto nel paragrafo precedente a proposito dell'implementazione della gestione dei sensori. Nella parte alta dello schema, compare, sotto forma di blocco funzionale, il circuito di gestione del motore della mano, come già descritto nell'articolo del mese scorso: non ci dilunghiamo oltre sull'argomento, invitando il lettore a consultare, per maggiori dettagli, la descrizione dello schema elettrico corrispondente. Sul lato sinistro dello schema si possono notare le connessioni alle linee della porta parallela, o dell'apparecchiatura *'PC Explorer'*, sulla quale è possibile avere maggiori informazioni visitando il sito: *'http://web.tiscali.it/spuntosoft/'*. Nella parte centrale sono visibili i sensori di pressione, che vengono utilizzati per conferire il 'tatto' alla nostra mano; sono collegati alle connessioni normalmente chiuse dei microinterruttori, per migliorarne la sensibilità attraverso tre resistenze di *'pullup'*. Il sensore di prossimità visibile, sul lato destro, è costituito da una coppia diodo LED-Fototransistor ad infrarossi, che costituiscono una barriera IR tra le dita della mano, attraverso la quale è possibile rilevare la presenza di un oggetto all'interno della pinza. L'impulso IR viene generato dal LED IR, su comando della linea *D2* della porta parallela e viene ricevuto attraverso l'ingresso della porta di status *S6.* Vale la pena spendere poche parole sulla possibilità di espansione del circuito: la configurazione volutamente mol-

Sistemi per montaggi sperimentali

Il sistema proposto in queste pagine è stato realizzato e collaudato con la apparecchiatura per il collaudo e la sperimentazione di circuiti elettronici con Personal Computer *'PC EXPLORER light':* ulteriori informazioni su come si possa reperire questa apparecchiatura è possibile visitare il WEB all'indirizzo:

http://web.tiscali.it/spuntosoft/

oppure inviare una e-mail a:

luca.spuntoni@ioprogrammo.it

Elettronica

Mano
meccanica: i sensori del tatto

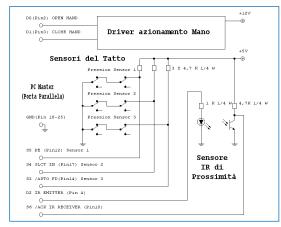



Fig. 6: **Lo schema elettrico del circuito necessario al controllo dei sensori della mano meccanica è stato inserito, per comodità del lettore nel file incluso al CD con il nome: "Schema_Elettrico_Sensori_Braccio_Meccanico.bmp".**

Fig. 8: **La realizzazione del circuito si riconduce alla connessione del cavo proveniente dalla mano meccanica, come mostrato in figura.**

to semplice lascia aperte svariate possibilità di sperimentazione. L'architettura del sistema rende possibile il massimo controllo software del braccio meccanico, rendendo disponibili tutti i segnali di uscita ed ingresso per la sua gestione; è possibile, comunque, automatizzare alcune funzioni di 'routine' per mezzo di alcuni circuiti elettronici complementari, che alleggeriscono il software di controllo, come vedremo negli appuntamenti futuri.

LA REALIZZAZIONE DEL CIRCUITO

La realizzazione del circuito proposto in questa sede è veramente molto semplice: lo schema elettrico si presta, inoltre, ad essere adattato ad altre applicazioni di controllo in cui si renda necessario l'utilizzo di microinterruttori e barriere a raggi infrarossi, come ad esempio nel controllo dei cancelli automatici. Il montaggio della parte relativa alla gestione del motore della mano meccanica è stato pubblicato sul numero di ioProgrammo precedente, per comodità del lettore se ne riporta comunque una immagine.

Il cablaggio può essere eseguito facilmente utilizzando l'apparecchiatura mostrata in Fig. 8, chiamata *'PC Explorer light'*, la più semplice della famiglia *'PC Explorer'*, sulla quale è possibile avere maggiori informazioni sul sito *'http://web.tiscali.it/spuntosoft/'*.

In alternativa, è possibile utilizzare le tecniche costruttive convenzionali, ovvero dotandosi di stagno, saldatore ed una buona dose di pazienza: il lettore ha in ogni caso, tutte le informazioni necessarie alla realizzazione della parte hardware e più avanti troverà il software di gestione, completo di componenti pronti all'uso, del programma compilato e funzionante dotato di codice sorgente.

Precauzioni

Prima di collegare il circuito al nostro PC occorre verificare la nostra realizzazione con attenzione, per assicurarci che tutto sia stato collegato come previsto.
L'utilizzo del programma presentato in questa sede, mentre è collegata una qualunque altra periferica al PC sulla porta LPT1, può bloccarne il funzionamento.

Fig. 7: **Nell'immagine si riporta il particolare costruttivo relativo al circuito di gestione del motore della mano meccanica: per maggiori dettagli si invita il lettore a consultare l'articolo relativo pubblicato sulla rivista del mese scorso.**

Fig. 9: **Nell'immagine si nota un particolare delle connessioni provenienti dai sensori della mano meccanica.**

Il cablaggio del circuito si riduce, in effetti, alla connessione delle linee del cavo provenienti dalla mano meccanica, alle connessioni corrispondenti della porta parallela, come è evidenziato nelle immagini qui incluse.

IL SOFTWARE DI CONTROLLO

Il programma discusso di seguito è contenuto nel CD allegato alla rivista, completo di codice sorgente e file eseguibile, per motivi di brevità ne verranno analizzate solamente le parti fondamentali alla comprensione della gestione software dei sensori della mano meccanica: per quanto riguarda il controllo del motore di azionamento della pinza, si invita il lettore a consultare l'articolo relativo pubblicato sul numero precedente. L'intestazione e la clausola *Uses*, in particolare, danno un'idea dei componenti esterni al programma

Fig. 10: Il programma di gestione dei sensori è in grado di controllare la presenza di un oggetto all'interno della mano e controllarne la presa.

che sono necessari al suo funzionamento: oltre a quelli standard di Delphi, possiamo notare *SpuntoLedComponent* e *UnitPortaParallela*, che rispettivamente inglobano la gestione grafica dei LED e tutta la gestione anche, a livello hardware, della porta parallela.
Tra le variabili pubbliche, della classe principale troviamo *ParallelPortDataAddress* e *ParallelPortStatusAddress*, che contengono gli indirizzi fisici delle 'porte' corrispondenti alla porta parallela: per ulteriori informazioni sullo standard Centronics si faccia riferimento all'articolo dello stesso autore: *'Controlliamo la porta Parallela con Delphi 6'*, pubblicato su ioProgrammo n. 57-58 Aprile e Maggio 2002. Le variabili pubbliche: *Sensor1Status*, *Sensor2Status*, *Sensor3Status* e *IRSensorStatus* conterranno lo stato logico dei rispettivi sensori. Alla creazione della finestra principale, viene eseguita la procedura *FormCreate*, nella quale vengono inizializzati tutti i parametri fondamentali per la corretta esecuzione del programma. In particolare è importante notare, l'inizializzazione delle variabili: *ParallelPortDataAddress* e *ParallelPortStatusAddress*, impostate per default sui valori di *LPT1*, per cui, se il lettore ha intenzione di utilizzare una porta diversa, dovrà impostare queste variabili in modo congruente con gli indirizzi fisici del suo sistema. La procedura *HandAction* è il cuore della gestione dei sensori della mano, oltre che del movimento della pinza. Il programma permette di muovere la pinza, indipendentemente dallo stato dei sensori *(Unconditioned Movement)*, in fase di posizionamento del braccio: è possibile però operare la chiusura della mano, fino a quan-

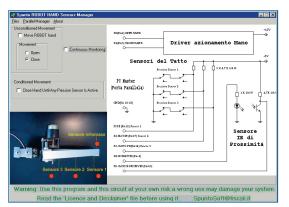



Fig. 11: Nell'immagine di figura si nota che tutti i LED relativi ai sensori della mano sono attivati: questo significa che la pinza preme l'oggetto in modo uniforme e che questo copre completamente l'interno della mano, dal momento che il sensore IR è a livello logico 'High'.

do uno dei sensori di pressione viene attivato, determinando la presa dell'oggetto *(Conditioned Movement)*. L'algoritmo di controllo è molto semplice e necessita di essere integrato con il movimento del braccio, per operare una strategia di manipolazione degli oggetti efficace. L'analisi della procedura è abbastanza evidente, essendo ancora ad alto livello, il movimento della mano si basa sulle procedure *OpenHand*, *CloseHand* e *StopHand*, mentre la lettura dello stato dei sensori avviene per mezzo di *ReadHandSensorsStatus*. La funzione *IsAnyPressionSensorPressed* verifica se anche uno solo dei sensori di pressione è stato premuto. Il timer *DelaytoStepTimer* si occupa di chiamare, ad intervalli regolari, la procedura *HandAction*, per forzarne l'esecuzione e tenere sempre aggiornato lo stato dei sensori e gestire il movimento della mano a seconda degli ordini impartiti dall'operatore. Per verificare lo stato logico dei sensori viene utilizzata la procedura *ReadHandSensorsStatus*, che si occupa, sensore per sensore, di leggerne lo stato logico, a livello di bit fisico, della porta relativa, impostare la variabile globale corrispondente al valore logico appropriato e accendere, oppure spegnere, il LED corrispondente in modo opportuno. Una particolare attenzione va dedicata al sensore infrarosso, infatti si opera per prima cosa l'in-

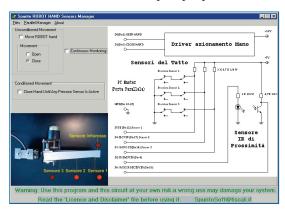



Fig. 12: In questo caso la pinza sta manipolando l'oggetto in modo tale da prenderlo con la parte anteriore: si notano i sensori 1 e 2 attivi e quello IR è a livello logico 'Low'.

Elettronica

Mano
meccanica: i sensori del tatto

Il software

Il software di controllo è stato collaudato con Win 3.x, Win 9x e Win Me, se si utilizza Win 2000, oppure NT, occorre scrivere una parte di codice che gestisca i privilegi del sistema, per non incorrere ad un errore del tipo '*Privileged error*'.

Elettronica

Mano *meccanica: i sensori del tatto*

vio di un impulso IR attraverso il bit *D3*, per poi verificarne la ricezione da parte del sensore IR attraverso la lettura dello stato logico del bit *S6*. La funzione *IsAnyPressionSensorPressed* restituisce *'True'* se almeno uno dei sensori di pressione viene premuto. Tre procedure si occupano della apertura della mano *(OpenHand)*, della chiusura *(CloseHand)* e dell'arresto del suo movimento *(StopHand)*. Infine, ancora a più basso livello, troviamo le procedure *WritePort* e *ReadPort* , che gestiscono rispettivamente l'invio all'indirizzo fisico *PortAddress* del valore contenuto in *PortData* e la lettura dello stato logico della porta di I/O indirizzata da *PortAddress*. Il programma Delphi, descritto in precedenza, ha la caratteristica di accedere all'hardware del PC attraverso i propri indirizzi fisici di I/O, questa tecnica, dal momento che scavalca il sistema operativo, potrebbe non 'piacere' a Windows NT, 2000, oppure XP, pertanto si consiglia di utilizzare un calcolatore dotato di Win 3.X, Win 9X, oppure Millennium. In alternativa occorre scrivere una parte di codice che gestisca i privilegi del sistema, per non incorrere ad un errore del tipo *'Privileged error'*. Una ulteriore alternativa che risolve ogni problema è la scrittura di un appropriato *'Device Driver'*, che però esula dallo scopo di queste pagine, data la complessità dell'argomento.

COLLAUDO DEL SISTEMA

Prima di collegare il circuito al nostro PC, occorre verificare la nostra realizzazione con attenzione, per assicurarci che tutto sia stato connesso come previsto: controlliamo che i connettori siano ben serrati e che nessuna parte metallica della mano possa urtare il circuito elettrico e creare cortocircuiti. Colleghiamo al nostro circuito il cavo relativo alla porta parallela del PC, se possediamo PC Explorer come mostrato in figura, oppure provvedendo a costruirci un cavo seguendo lo schema elettrico e la Tab. 1 riportati all'inizio dell'articolo. Lanciamo il programma e proviamo a fare aprire e chiudere la mano, poi verifichiamo l'effettivo funzionamento dei sensori, premendoli uno per uno e confrontiamone il variare dello stato logico sul display del programma. Verifichiamo il funzionamento della barriera a raggi infrarossi interrompendone il flusso IR e verificandone la variazione dello stato di accensione del LED relativo. Se il circuito non funziona, provvediamo a spegnere tutto prima di ricontrollare i collegamenti e riprovare di nuovo.

La nostra applicazione di controllo dei sensori della mano meccanica è a questo punto terminata: disponiamo di un sistema che è in grado di azionare la pinza, controllarne la presa per mezzo di sensori di pressione e verificare il corretto posizionamento dell'oggetto per mezzo del sensore a raggi infrarossi.

Fig. 13: Dopo avere terminato l'assemblaggio del circuito, siamo pronti a collegare un comune cavo parallelo alla nostra apparecchiatura, oppure nel caso in cui il circuito sia stato autocostruito siamo pronti a collegarlo alla porta parallela del PC.

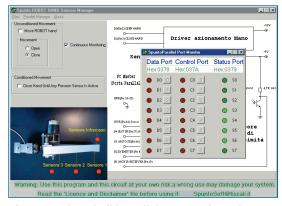



Fig. 14: Attivando il 'Parallel Manager' possiamo verificare in qualunque momento lo stato logico delle Porte Dati, di Status e di Controllo della Porta parallela: gli indirizzi di default sono quelli di LPT1.

CONCLUSIONI

In queste pagine abbiamo visto come realizzare un controllo sensoriale tattile per il nostro braccio meccanico: il progetto dello schema elettrico, tutti i collegamenti necessari, il software compilato ed i relativi codici sorgenti sono stati messi a completa disposizione del lettore. Gli algoritmi di manipolazione degli oggetti verranno sviluppati nei prossimi appuntamenti, in modo tale da conferire potenza e versatilità alla nostra applicazione. In particolare, nel prossimo articolo analizzeremo la gestione del secondo grado di libertà del braccio meccanico, relativo alla rotazione del polso. Il lettore vorrà comprendere che nonostante quanto esposto in queste pagine sia stato debitamente verificato e collaudato, tuttavia viene riportato a scopo illustrativo e di studio, pertanto l'editore e l'autore non sono da considerare responsabili per eventuali conseguenze derivanti dell'utilizzo di quanto esposto in questa sede, soprattutto per la tipologia e la complessità dell'argomento. Per maggiori informazioni sul braccio meccanico, sulle interfacce relative e sull'apparecchiatura 'PC Explorer light' è possibile visitare il sito: *'http://web.tiscali.it/spuntosoft/'*, inoltre l'autore è lieto di rispondere ad ogni richiesta di chiarimento o delucidazione sull'argomento.

Luca Spuntoni

Bibliografia

• CONTROLLIAMO LA PORTA PARALLELA CON DELPHI 6
Luca Spuntoni
ioProgrammo N° 57-58
Aprile e Maggio 2002

☑ Interazione fra Java e Office.

Generare un file Excel in Java

Sistema

In quest'articolo vedremo come creare in Java dei fogli Excel 97/XP utilizzando le librerie JExcelAPI di Andy Khan.

Quante volte ci è capitato che, dopo aver scritto la nostra fantastica applicazione Java, la presentiamo e il commento più frequente è: "Bello, ma questi dati, poi, come posso usarli altrove?". E mentre rispondiamo che si potrebbe creare un file xml o un file ascii separato da virgola, ecc., ci sentiamo dire: "ma come file excel, no?". Bene, grazie alle classi *JExcelAPI*, scritte da Andy Khan, abbiamo la possibilità di creare facilmente dei file *xls* completi con diversi fogli. In questo articolo daremo prima velocemente una definizione di un file excel, vedremo come utilizzare le classi *JExcelApi* per leggere e scrivere un foglio excel e infine realizzeremo una classe che ci permetterà di generare un file excel da una lista di *Jtable*.

COS'È UN FOGLIO EXCEL?

Un file Excel 97/XP non è un unico foglio elettronico, bensì una cartella di lavoro (*Workbook*) che può contenere più fogli (*SpreadSheet*). Ogni foglio è una tabella, in cui ogni cella contiene un dato (testuale, numerico o data, secondi diversi formati) oppure, e qui sta la differenza tra un foglio elettronico e una semplice tabella in word o html, una formula, ovverosia una serie di operazioni che portano ad un dato calcolato che può utilizzare valori presenti in altre celle (Fig.1). Una limitazione della libreria *JExcelAPI*, è quella di non poter scrivere formule nelle celle, ma solo dati, mentre è possibile leggerle. Nel caso di semplice esportazione dati questa non è una grossa limitazione, in quanto una volta che abbiamo i dati in formato Excel possiamo elaborarli a nostro piacimento, con tutti gli strumenti che un foglio elettronico ci mette a disposizione.

Fig. 1: **Editing di una formula in Excel XP.**

LEGGIAMO UN FOGLIO EXCEL

Per utilizzare le classi *JExcelAPI* assicuriamoci che nel nostro classpath sia presente il file *jxl.jar* e che tra i nostri import sia presente:

```
import jxl.*;
```

La cartella di lavoro Excel è rappresentata dalla classe *Workbook*. Per ottenere un'istanza di questa classe (da una cartella di lavoro Excel esistente) si usa il metodo statico *getWorkbook*, che presenta diverse implementazioni, per poter leggere da *File* o da un *InputStream*. Per creare un oggetto *Workbook* da un file di nome *fileName* scriviamo:

```
Workbook workbook = Workbook.getWorkbook(new
                                    File(fileName));
```

A questo punto dobbiamo accedere ai fogli di lavoro presenti in questo workbook. Per sapere quanti sono, possiamo utilizzare il metodo *getNumberOfSheets*. Il foglio di lavoro è rappresentato dalla classe Sheet. Per accedere all'i-esimo foglio del workbook usiamo il metodo *getSheet(i)*. Il seguente codice mostra come recuperare da un workbook i diversi fogli di lavoro:

```
int sheetCount = workbook.getNumberOfSheets();
/* ciclo per il numero di figli-fogli nella cartella excel */
for (int i=0; i<sheetCount; i++) {
   Sheet sheet = workbook.getSheet(i);
   System.out.println("Foglio "+i+": "+sheet.getName());
   ....
```

☑ CD ☑ WEB
JExcelTables.java
cdrom.ioprogrammo.it

Dove reperire JExcelApi

Le librerie *JExcelAPI*, scritte da Andy Khan, sono presenti sul CD allegato. Le più aggiornate sono scaricabili all'indirizzo:

http://www.andykhan.com/jexcelapi/download.html.

L'ultima versione è la 2.3.6. Le JExcelAPI sono open-source secondo la licenza GNU reperibile all'indirizzo:

http://www.gnu.org/copyleft/lesser.html

Sistema

Generare
un file Excel in Java

Precedenza di Import in Java

Tramite le istruzioni di import, il compilatore riesce a risolvere una classe senza dover specificare il suo package. Quando abbiamo due classi omonime in due package diversi, entrambi nell'import, scrivendo solo il nome della classe il compilatore non riesce a risolvere a quali delle due classi stiamo facendo riferimento e quindi la classe va scritta con il path completo del package. Tuttavia, se vogliamo che il nome della classe, senza package, sia riferito a una delle due, ripetiamo un'istruzione di import specifica per quella classe, dopo gli import di package. In questo modo, quando il compilatore troverà il nome della classe senza il package, sa che deve risolverla secondo l'ultimo import specificato.

```
    // leggi il foglio di lavoro
    ....
}
```

Il prossimo passo è accedere alle singole celle del foglio di lavoro. La classe che rappresenta la cella è la classe *Cell*. La singola cella si ottiene grazie al metodo *getCell* della classe *Sheet*. Essendo il foglio di lavoro una griglia (una tabella), devo specificare alla chiamata di questo metodo le coordinate della cella che mi interessa. La prima cella (che in Excel è *A1*) ha le coordinate (*0,0*). Quindi se voglio accedere alla cella *C4*, devo usare il seguente codice:

```
Cell c4 = sheet.getCell(2,3);
```

Per leggere il contenuto della cella posso chiamare il metodo *getContents*, che mi restituisce la conversione a stringa del contenuto della cella.

```
String sc4 = c4.getContents();
```

In questo modo stiamo ignorando il tipo di dato memorizzato nella cella (testo, numerico o data) e stiamo trattando tutto come stringa. Se il nostro scopo è solo quello di visualizzare i dati del foglio, questa soluzione ci può andare bene, ma se intendiamo eseguire delle elaborazioni sui dati che leggiamo dal foglio excel, allora trattare numeri e date come stringhe non può soddisfarci. In questo caso possiamo utilizzare le sottoclassi di *Cell* che rappresentano le specifiche celle per il testo (*LabelCell*), per i valori numerici (*NumberCell*) e per le date (*DateCell*). Per capire se una *Cell* è *LabelCell*, *NumberCell* o *DateCell* possiamo procedere con il più classico degli *instanceof* oppure utilizzare il metodo *getType()* che restituisce un *CellType* di tipo *Enumeration* nel seguente modo:

```
Cell c4 = sheet.getCell(2,3);
if (c4.getType() == CellType.LABEL) {
  String sc4 = ( (LabelCell)c4 ).getString();
  /* uso la stringa */
}
else if (c4.getType()==CellType.NUMBER) {
  double nc4 = ( (NumberCell)c4 ).getValue();
  /* uso il numero */
}
else if (c4.getType()==CellType.DATE) {
   Date dc4 = ( (DateCell)c4 ).getDate();
   /* uso la data */
}
```

Una volta che la nostra lettura della cartella di lavoro Excel è terminata, è bene chiamare il metodo *close* per liberare memoria e risorse:

```
workbook.close();
```

Vediamo ora come poter creare cartelle di lavoro Excel.

SCRIVIAMO UN FOGLIO EXCEL

Per usare le librerie di classi *JExcelApi* necessari per scrivere cartelle di lavoro Excel, abbiamo bisogno, rispetto alla sola lettura, di importare anche il package *jxl.write*:

```
import jxl.*;
import jxl.write.*;
```

Per creare una cartella di lavoro Excel nuova usiamo il metodo statico della classe *Workbook createWorkbook*. Questo metodo presenta diverse implementazioni per creare la cartella Excel in un file o in maniera più generale su un *OutputSream*. Questa possibilità può risultare veramente utile nelle applicazioni web, in quanto passando come *outputStream* un *ServletOutputStream*, possiamo fare in modo che la servlet possa creare una cartella Excel da mandare al browser del client (senza necessità di creare un file sul server) che potrà essere salvato o aperto sul client come un normale file excel. Il metodo *createWorkbook* restituisce un oggetto di tipo *WritableWorkbook*, sottoclasse di *Workbook*, a cui aggiunge le funzionaltà per la creazione e la scrittura di fogli di lavoro. L'istruzione per creare una cartella excel in un file *fileName* è la seguente:

```
WritableWorkbook workbook =
        Workbook.createWorkbook(new File(fileName));
```

Adesso dobbiamo creare i singoli fogli di lavoro. Per far ciò utilizziamo il metodo *createSheet* (proprio della classe *WritableWorkbook*) che prevede come parametri il nome del foglio di lavoro e la sua posizione (posto che la posizione 0 è la prima). Analogamente a quanto successo per il metodo *createWorkbook*, anche il metodo *createSheet* restituisce una sottoclasse di *Sheet*, *WritableSheet*, che appunto aggiunge le funzionalità per la scrittura di dati nel foglio.

```
WritableSheet sheet = workbook.createSheet("Primo
                                        Foglio",0);
```

Non rimane altro, a questo punto, che aggiungere i dati nel foglio di lavoro, ovverosia creare celle e aggiungerle al foglio nella posizione a noi gradita.
Per aggiungere le celle si usa il metodo *addCell* che accetta come argomento oggetti che implementano l'interfaccia *WritableCell* e tra queste troviamo *Number* (per inserire numeri nella cella), *Label* (per inserire stringhe) *DateTime* (per inserire date e orari) e *Boolean* (per inserire valori *true/false*).
Vogliamo fare a questo punto due osservazioni:

- si noti come la posizione della cella non venga impostata alla chiamata del metodo *addCell*, bensì nel costruttore delle singole celle *Number*, *Label*, *Boolean*, *DateTime* nel formato colonna, riga

partendo in entrambi i casi da 0;

- i nomi delle classi *Number, Label, Boolean* sono già presenti nella gerarchia delle classi di Java. Quando le utilizzate insieme ad altri *package Java* che contengono i loro omonimi (*java.awt, java.-lang*, che peraltro è sempre importato di default) il compilatore non sa risolvere il nome. In questi casi potete utilizzare il nome completo della classe (comprensivo di package) e queste classi si trovano nel package *jxl.write*. Quindi se volete usare la classe *Number* scrivete *jxl.write.Number*.

Chiariti questi due punti vediamo come inserire nel foglio di lavoro una stringa, un numero, una data e un valore booleano nella prima riga:

```
jxl.write.Number n = new jxl.write.Number(0,0, 17.2);
Label l = new Label (1,0, "Prova Stringa");
DateTime dt = new DateTime(2,0, new Date());
jxl.write.Boolean b = new jxl.write.Boolean(3,0, true);
```

Una volta completate le operazioni di aggiunta celle per tutti i fogli di lavoro, si deve chiamare il metodo *write* della classe *WritableWorkbook* per scrivere effettivamente le operazioni effettuate. Solo dopo la chiamata al metodo *write* si può procedere alla chiusura del workbook tramite il metodo *close*.

```
workbook.write();
workbook.close();
```

Una chiamata al metodo *close*, senza una precedente al metodo *write*, porterebbe alla generazione di un file completamente vuoto.

DA JTABLE A FOGLIO DI LAVORO EXCEL

Fino ad ora abbiamo visto, per sommi capi, come funzionano le librerie *JExcelApi*. Ora vediamo di applicarle per creare qualcosa che, nella migliore delle tradizioni dei linguaggi a oggetti, sia riusabile e generale. Ci proponiamo di scrivere una classe che, data una lista di *JTable*, crei una cartella di lavoro Excel, dove ad ogni *JTable* corrisponde un foglio di lavoro Excel.
Chiamiamo la nostra classe *JExcelTables* e predisponiamo tutti gli import che ci servono:

```
import jxl.*;
import jxl.write.*;
import java.util.*;
import javax.swing.*;
import java.io.*;
import java.lang.Boolean;
import java.lang.Number;
```

Notate che abbiamo esplicitato l'import di *Boolean* e *Number* del package *java.lang* che normalmente non è necessario. Per la spiegazione rimando al box relativo agli *import* in Java. Come variabili di classe abbiamo due liste: una per le tabelle che vogliamo inserire nella cartella di lavoro Excel (ogni elemento della lista sarà una *JTable*) e una per i nomi che vogliamo dare ad ogni foglio di lavoro Excel corrispondente alla *JTable* (ogni elemento della lista sarà una *String*). Entrambe le liste vengono inizializzate come *ArrayList* vuote:

```
List tables = new ArrayList();
List tableNames = new ArrayList();
```

Definiamo due costruttori. Il primo prende una stringa ed una tabella e le aggiunge alla lista dei nomi delle tabelle e delle tabelle. In molti casi infatti il nostro file Excel sarà costituito da una sola tabella e vogliamo che da fuori si possa usare un costruttore semplice come questo senza scomodare le *List*.

```
public JExcelTables(String name, JTable table) {
   tables.add(table);
   tableNames.add(name);
}
```

Il secondo costruttore è più generico e prevede una lista per i nomi delle tabelle e una per le tabelle.
Il codice non effettua alcun controllo ma è chiaro che la dimensione delle due liste deve essere la stessa (altrimenti fioccherano le *ArrayIndexOutOfBoundsException*) e che la lista *tables* deve contenere solo oggetti di tipo *JTable* (altrimenti in questo caso voleranno molte *ClassCastException*).

```
public JExcelTables(List tableNames, List tables)
{
          this.tableNames.addAll(tableNames);
          this.tables.addAll(tables);
}
```

Ora passiamo alla stesura del metodo più importante, quello che legge dalle tabelle e scrive sul file Excel. Chiamiamo il metodo *createExcelWorkbook* e come parametro gli passiamo un *OutputStream* su cui andare a creare il workbook.

```
public void createExcelWorkbook(OutputStream out)
                                throws Exception
{
   ....
```

Per prima cosa creiamo il *WritableWorkbook*.

```
WritableWorkbook workbook =
                        Workbook.createWorkbook(out);
```

Per ogni tabella creiamo un foglio di lavoro con il nome che prendiamo dalla lista *tableNames*.

Sistema

Generare
un file Excel in Java

Software di esempio

Il codice sorgente della classe *JExcelTables* è nel package di default. Basta compilarla, avendo il file *jxl.jar* nel classpath. Per provare l'esempio lanciarla come applicazione.

Sistema

Generare
un file Excel in Java

```
for (int i = 0; i < tables.size(); i++)
{
  JTable table = (JTable)tables.get(i);
  /* per ogni tabella creiamo un foglio di lavoro */
  WritableSheet sheet = workbook.createSheet(
                              tableNames.get(i).toString(),i);
  ...
```

Nella prima riga del foglio di lavoro andiamo a scrivere i nomi delle colonne su uno sfondo grigio. Per specificare il coloro di sfondo della cella usiamo la classe *WritableCellFormat* e il suo metodo *setBackground*.

```
  ...
  /* nella prima riga scriviamo i nomi delle colonne */
  /* Scriviamo i nomi delle colonne con uno sfondo
                                          grigino */
  WritableCellFormat cellFormat = new
                                 WritableCellFormat();
  cellFormat.setBackground(jxl.write.Colour.GRAY_25);
  for (int j = 0; j < table.getColumnCount(); j++)
  {
      String colname = table.getColumnName(j);
      sheet.addCell(new Label(j,0,colname,cellFormat));
  }
  ...
```

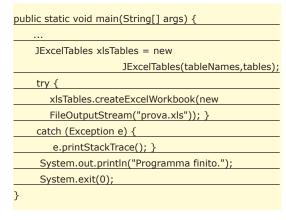
Ora scorriamo la tabella (prima per colonna e poi per riga) e, a seconda del tipo dell'oggetto che troviamo alle coordinate in questione, creiamo la relativa cella da aggiungere al foglio di lavoro. Notate come l'indice della riga nel foglio di lavoro sia aumentato di 1 in quanto nella prima riga c'è l'intestazione con i nomi delle colonne.

```
  /* Ora scorriamo la tabella e inseriamo le celle */
  for (int j = 0; j < table.getColumnCount(); j++) {
    for (int k = 0; k < table.getRowCount(); k++) {
       Object datum = table.getValueAt(k,j);
       if (datum instanceof Number) {
          Number ndatum = (Number)datum;
          jxl.write.Number number = new
          jxl.write.Number(j,k+1,ndatum.doubleValue());
          sheet.addCell(number);
       }
       else if (datum instanceof Date) {
          Date ddatum = (Date)datum;
          DateTime date = new DateTime
                                      (j,k+1,ddatum);
          sheet.addCell(date);
       }
       else if (datum instanceof Boolean) {
          Boolean bdatum = (Boolean)datum;
          jxl.write.Boolean bool = new
          jxl.write.Boolean(j,k+1,
                              bdatum.booleanValue());
          sheet.addCell(bool);
       }
       else {
          String sdatum = datum.toString();
          Label label = new Label(j,k+1,sdatum);
          sheet.addCell(label); }
       }
    }
}
```

Infine rendiamo effettive le modifiche con la chiamata a *write* e chiudiamo il workbook.

```
workbook.write();
workbook.close();
```

Ora nel metodo *main* scriviamo del codice per testare la nostra classe.

```
public static void main(String[] args) {
   ...
   JExcelTables xlsTables = new
                     JExcelTables(tableNames,tables);
   try {
      xlsTables.createExcelWorkbook(new
      FileOutputStream("prova.xls")); }
   catch (Exception e) {
      e.printStackTrace(); }
   System.out.println("Programma finito.");
   System.exit(0);
}
```

Se tutto è andato bene, viene creato il file *prova.xls* che, una volta aperto, dovrebbe apparire come in Fig. 2.

Nome	Cognome	Data di nascita	altezza
Marco	De Marchi	21/03/1954	1,67
Miriam	Polloni	17/01/1977	1,54
Elvio	Murlacci	11/07/1962	1,84

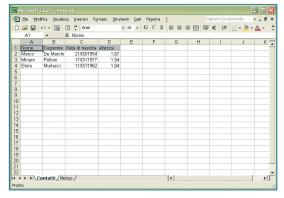



Fig. 2: **Il file da noi creato *prova.xls* come appare aperto da Excel XP.**

CONCLUSIONI

In questo articolo abbiamo visto come poter leggere e scrivere file Excel utilizzando le librerie *JExcelAPI*. Abbiamo mostrato l'implementazione della classe *JExcelTables* per creare una cartella di lavoro Excel da una lista di *JTable*, dimostrando l'utilità delle librerie *JExcelAPI* nello sviluppo di codice riusabile per l'esportazioni di dati da applicazioni Java (stand-alone o web) verso il mondo office.

Marcello Valeri

L'autore

Marcello Valeri è laureato in Informatica e lavora a Roma come informatico presso il Ministero degli Esteri. I suoi interessi sono orientati all'accesso remoto a database. Può essere contattato attraverso la redazione di ioProgrammo.

☑ Delphi, la localizzazione del software.

La tua applicazione in tutte le lingue

Sistema

La globalizzazione influenza noi programmatori molto più di quanto si possa immaginare. Imprese, mercati e comunità di ogni genere operano correntemente in una dimensione mondiale, superando quindi i confini dei singoli stati, perché non dovrebbero farlo anche i nostri programmi? In questo articolo analizzeremo i problemi ed i vantaggi della localizzazione.

L'utente medio si aspetta che qualsiasi software "parli" nella propria lingua, non tanto per motivi di orgoglio nazionale quanto per condivisibili ragioni di produttività. E' esperienza comune che, operare con applicazioni tradotte in italiano significhi, per noi, acquisire padronanza in tempi più brevi, diminuendo così frustrazioni ed errori. La versione localizzata, una volta prerogativa dei programmi più blasonati, costosi e diffusi, sta rapidamente diventando una moda o forse una necessità. Ormai, grazie a Internet, anche piccoli programmi gratuiti o shareware vengono distribuiti in più lingue, spesso per merito dell'opera di traduttori volontari. Per progetti di grandi dimensioni solitamente si preferisce ricorrere a società specializzate. Prima di proseguire è opportuno definire la differenza tra internazionalizzazione (anche nota come globalizzazione) e localizzazione. La localizzazione è il processo di adattamento di un software alle esigenze degli utenti di un determinato paese. La semplice traduzione nella maggior parte dei casi non è sufficiente e deve essere accompagnata dall'adeguamento dell'applicazione ai costumi locali e alle leggi vigenti. L'internazionalizzazione è un concetto a più ampio respiro in quanto include tutte le decisioni tecniche, progettuali, manageriali, finanziarie e di marketing che consentono ed agevolano la fase di localizzazione. Internazionalizzazione e localizzazione sono gergalmente abbreviate con *I18N* e *L10N*: si considerano le lettere iniziali e finali dei termini inglesi *InternationalizatioN* e *LocalizatioN* separate dal numero di lettere intermedie.

LINEE GUIDA

Il processo di localizzazione richiede un'attenta pianificazione, i fattori da considerare sono molti, ed ognuno può minare il successo di un software rispetto a prodotti concorrenti. Tra programmi con funzionalità simili, il famigerato "utente medio" tende a scartare quelli non disponibili nella propria lingua e quelli localizzati in modo poco professionale. Vediamo quali aspetti non bisogna assolutamente sottovalutare se le dimensioni del progetto non sono tali da richiedere il servizio di società specializzate nella localizzazione:

- **Interfaccia** – Il primo impatto dell'utente è con l'interfaccia del programma, chiaramente la traduzione deve risultare corretta ed impiegare i termini più idonei. La tecnica preferita dalla maggior parte dei programmatori consiste nell'inserire le stringhe in file di risorse esterni e non direttamente nel codice sorgente. Tali risorse possono essere distribuite ai traduttori madrelingua e successivamente integrate nell'applicazione come testo tradotto. Vedremo un paio di esempi in seguito. Nella progettazione dell'interfaccia dobbiamo anche pensare che la lunghezza delle parole varia da lingua a lingua, dunque è opportuno lasciare spazio sufficiente tra i vari controlli visuali. Un procedimento euristico suggerisce di realizzare la GUI *(Graphical User Interface)* in inglese per poi valutare un incremento della lunghezza delle stringhe non inferiore al 30% (Fig.1).

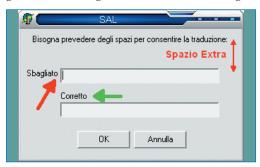



Fig. 1: Lasciamo più spazio possibile per il testo tradotto.

Unicode

Gli 8 bit del set ASCII non sono sempre sufficienti per mappare tutti i caratteri di un alfabeto. Per questo motivo il consorzio Unicode Inc ha progettato e mantiene un set di caratteri universale a 16 bit. Con 16 bit è possibile, nella pratica, codificare i caratteri di ogni alfabeto.

Sistema

La tua *applicazione in tutte le lingue*

Formati

Ogni paese usa una propria combinazione di formati, chiamata in inglese *locale*, per memorizzare numeri, date, valute, numeri di telefono, indirizzi, etc. Conviene perciò avviare la localizzazione durante la fase progettuale riducendo così complessità del codice, inconvenienti, ritardi e costi. Ponetevi per esempio il problema di gestire la data 18 maggio 1977, vista come 1977/05/18 da un giapponese, 05/18/1977 da un americano, 18.05. 1977 da un tedesco e così via…

HTML

Internazionalizzare siti Web è certamente un'operazione difficile. Il processo è problematico anche perchè il linguaggio HTML è stato progettato basandosi sul set di caratteri ISO 8859-1 (noto come ISO Latin-1). Fortunatamente il set ISO 10646, contenente alcune decine di migliaia di caratteri,si sta affermando come nuovo standard.
Un formato di trasformazione dell'ISO 10646 è il famoso UTF-8.

- **Documentazione** – La traduzione della documentazione, cartacea o in formato elettronico, richiede tempi e costi non trascurabili, ecco perché non di rado le applicazioni meno costose sono fornite con i manuali nella sola lingua originale.
- **Grafica** – Immagini e icone contribuiscono a rendere amichevole ed accattivante un'interfaccia ma potrebbero anche causare seri problemi per via delle diversità culturali. Alcune metafore e simboli non trovano un equivalente in altre culture, gli effetti del loro utilizzo possono variare dal semplice disorientamento dell'utente fino all'offesa esplicita. Inoltre, per favorire la traduzione dei testi presenti nelle immagini, si preferisce impiegare formati grafici, ad esempio il PSD di Adobe Photoshop, che supportino i livelli indipendenti (Fig.2).

Fig. 2: Grafica e testo devono trovarsi in due livelli diversi,un formato comune è il PSD di Photoshop.

Il sistema operativo Windows mette a disposizione del programmatore una serie di funzioni per gestire: impostazioni locali, set di caratteri e input da tastiere internazionali. Vi consiglio di dare loro quanto meno un'occhiata consultando la documentazione a corredo dei vari Platform SDK. Nel seguito dell'articolo esamineremo alcune possibili tecniche di separazione delle stringhe dal codice: *Integrated Translation Environment (ITE), GNU GetText e assembly satellite*. In generale, le stringhe vengono archiviate in uno o più file chiamati, in base al metodo di separazione e del linguaggio utilizzato, *tabelle di stringhe, file di messaggi, file di proprietà*. Dopo la compilazione il programma può vedere il testo tradotto come librerie DLL caricate dinamicamente, normali file di testo o risorse dell'eseguibile a seconda dei casi. Analizzeremo tre possibili soluzioni: la prima incentrata su Delphi, la seconda di taglio generale e la terza orientata alla piattaforma .NET.

INTEGRATED TRANSLATION ENVIRONMENT (ITE)

Le ultime versioni dell'ambiente di sviluppo Delphi contengono numerosi strumenti che semplificano il lavoro di programmatori e traduttori, si va dall'ITE alle classi per il controllo completo di locale, formati data/ora, set di caratteri e metodi di input da tastiere asiatiche. Concentreremo la nostra attenzione sui programmi che ci consentono di localizzare con metodologie RAD un'applicazione per mezzo di DLL di risorse. I translation tools sono essenzialmente tre:

- **Translation Manager** – mostra una griglia per la visualizzazione e la modifica delle stringhe presenti nel programma da localizzare. Il *Translation Manager* è accessibile dal menu *View* | *Translation Manager* di Delphi (Fig.3).

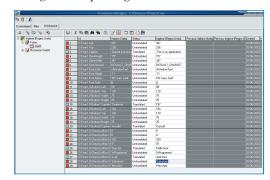



Fig. 3: Il Translation Manager in tutto il suo splendore, gestisce le traduzioni in qualsiasi lingua.

- **Translation Repository** – è un archivio con estensione *.rps* che memorizza traduzioni condivise da più progetti.
- **Resource DLL Wizard** – genera e gestisce le DLL di risorse, una per ogni lingua aggiunta al progetto. Le risorse sono salvate in formato testuale in file *.rc*.

Per aggiungere una nuova lingua alla nostra applicazione dobbiamo solo cliccare sul menu *Project* | *Languages* | *Add* e selezionarla dalla lista. Dopo aver confermato le scelte proposte siamo pronti a tradurre le varie stringhe (Fig. 3), aggiornare le DLL di risorse e compilare il programma. Otterremo un eseguibile capace di adattarsi automaticamente alla lingua del sistema operativo (Fig. 4) grazie alle DLL create durante la compilazione (le librerie a collegamento dinamico assumono come estensione le tre lettere della lingua, ad esempio *nomeprogramma. ITA, nomeprogramma.ENG*, …). Se vogliamo testare il risultato, prima dell'esecuzione modifichiamo la lingua corrente con l'opzione *Project* | *Languages* | *Set Active*. Naturalmente perché tutto funzioni le DLL, risultanti dal processo di localizzazione, devono essere distribuite insieme all'eseguibile. L'*Integrated Translation Environment* di Delphi meriterebbe una trattazione senz'altro più dettagliata ma vorrei proporvi una soluzione alternativa altrettanto potente e soprattutto valida in un contesto più generale.

GNU GETTEXT

Come al solito ioProgrammo pensa anche al pro-

Sistema

La tua applicazione in tutte le lingue

Fig. 4: L'applicazione in italiano e quella in inglese a confronto, non è necessario ricompilare: il programma si adatta automaticamente alla lingua del sistema.

grammatore che non vuole spendere una fortuna per procurarsi gli strumenti migliori, purtroppo non sempre alla portata di tutte le tasche. Per questo motivo verrà presentato il progetto open-source *GNU GetText*, sicuramente noto agli smanettoni appassionati di Linux ma forse (ancora) poco conosciuto nel mondo Windows. La tecnologia è molto interessante perché, oltre ad essere completamente gratuita, viene impiegata con successo da migliaia di applicazioni scritte nei linguaggi più disparati: C/C++, Delphi, Python, Kylix, etc. Gli eseguibili possono addirittura inglobare più traduzioni, con pieno supporto Unicode, utilizzabili contemporaneamente e selezionabili in fase di esecuzione. Uno dei tanti punti di forza del pacchetto *GetText* è rappresentato dal fatto che il traduttore può vedere in tempo reale gli effetti del proprio lavoro senza dover necessariamente inviare i file al programmatore per la ricompilazione. La lista dettagliata delle caratteristiche è impressionante, per motivi di spazio ci limiteremo all'implementazione di un piccolo programma che supporti due lingue. Un programma basato su *GetText*, come default, verifica le impostazioni della lingua del sistema operativo e si comporta di conseguenza, è però possibile modificare la lingua con il comando *UseLanguage('it')* o impostando la variabile di ambiente *LANG* tramite *SET LANG=it*. Per una lista completa dei codici si consulti *http://www.loc.gov/standards/iso639-2/langcodes.html*. Dopo aver scaricato ed installato i pacchetti "GNU gettext per Delphi, C++ Builder e Kylix" e poEdit, il secondo non è strettamente necessario ma facilita la traduzione delle stringhe, siamo pronti per creare il nostro programma localizzato. I passi da seguire, dopo aver implementato il programma in lingua inglese, sono pochi:

1) Ogni **unit** contenente delle form deve avere un riferimento a *gnugettext* nella clausola *uses* e la chiamata *TranslateComponent(Self)* nel gestore di evento *OnFormCreate* delle varie form. L'operazione è piuttosto noiosa quindi vi consiglio di scaricare i componenti *"dxgettext helpers"* dal sito del progetto poiché automatizzano questa fase.
2) **poEdit** accetta in ingresso dei file *.PO (Portable Object)*, uno per ogni lingua da supportare, e produce file con estensione *.MO* (*Machine Object*). I primi sono normali file di testo modificabili dal

Fig. 5: Con un solo click si estraggono le stringhe dai sorgenti!

traduttore, i secondi sono binari interpretati dal programma localizzato. L'installazione di *GNU GetText* per Delphi aggiunge delle voci al menu contestuale di Explorer, in particolare rende disponibile il comando che estrae le stringhe da tradurre (Fig.5) e le salva in un file con estensione *.po*. Se non avete *poEdit* (Fig.6) potete comunque utilizzare un qualsiasi editor di testo per localizzare l'applicazione (il file di testo dovrebbe essere codificato con il set UTF-8) e successivamente creare il file *.mo* con il programma da linea di comando *msgfmt.exe*.

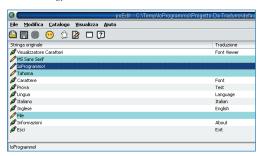



Fig. 6: poEdit ha funzionalità analoghe a quelle del Translation Manager di Delphi.

3) Il menu contestuale espanso da *GNU GetText* permette di inglobare le traduzioni nell'eseguibile, una scelta sicuramente consigliata in quanto ne semplifica la distribuzione! La voce *Embed Translations* non fa altro che richiamare il comando *ggassemble.exe*. Se invece preferite ridurre le dimensioni dell'eseguibile tramite compressori dovete posizionare in modo corretto i file .mo nella struttura di directory *locale\XX\LC_MESSAGES*, XX indica il codice della lingua (Fig.7).

Librerie a collegamento dinamico

Le librerie a collegamento dinamico hanno il grande vantaggio di essere aggiornabili separatamente rispetto al programma principale evitando così fastidiose ricompilazioni; il codice sorgente e l'eseguibile restano dunque inalterati indipendentemente dalle lingue aggiunte, eliminate o modificate.

Forme plurali

Molte lingue hanno più forme plurali, il pacchetto GNU GetText per risolvere il problema prevede la funzione *ngettext(EnglishSingularForm, EnglishPluralForm : string; Number : Longint).* La documentazione ne descrive l'utilizzo in modo esauriente.

Sistema

La tua *applicazione in tutte le lingue*

Fig. 7: I file .mo devono rispettare la struttura CartellaApplicazione\Locale\LC_MESSAGES.

Come risultato finale si ottiene un programma tradotto dinamicamente in inglese/italiano (Fig.8) a seconda delle scelte dell'utente; non è affatto difficile aggiungere il supporto per altre lingue. La porzione di codice interessante è riportata in seguito, sul cd allegato alla rivista e nella sezione CD-Rom sul Web trovate il programma ed il sorgente completo:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 TranslateComponent (self); // EFFETTUA LA TRADUZIONE
 Caratteri.Items.Assign(Screen.Fonts); // Ottieni la lista
                                                dei caratteri
 Caratteri.ItemIndex:=0; // Punta al primo
 Button1Click(Self); // Prova il carattere
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
 // Cambia il carattere della "stringa di prova"
 with Caratteri do
  Label1.Font.Name:=Items[ItemIndex];
end;
procedure TForm1.Esci1Click(Sender: TObject);
begin
 Close; // Esci dal programma
end;
procedure TForm1.Informazioni1Click(Sender: TObject);
begin
 MessageDlg(_('ioProgrammo - Salvatore Meschini'),
                      mtInformation,[mbOK],0); // Mostra info
end;
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
// Scelta dell'utente:
 case RadioGroup1.ItemIndex of
   0:UseLanguage('it');
   1:UseLanguage('en');
 end;
 TranslateComponent(self); // NUOVA TRADUZIONE
end;
```

Gnu GetText

La versione standard di GNU GetText è pienamente compatibile con programmi scritti in C/C++ (e non solo!), di conseguenza, nel caso in cui si prediligano gli altri linguaggi supportati, le tecniche finora descritte sono soggette solo a lievi modifiche.

.NET FRAMEWORK

La piattaforma .NET di Microsoft è un prodotto recente perciò comprende differenti tecnologie specificamente progettate per la localizzazione del software. Il namespace *System.Globalization*, a titolo di esempio, contiene le classi per la gestione della lingua, dei formati data/ora/valuta, dell'ordinamento delle stringhe, etc. Le risorse sono memorizzate in file XML di tipo RESX (Fig.9) e vengono caricate in base al valore della proprietà *CultureInfo.CurrentUICulture*. Tale proprietà può essere impostata nel codice dell'applicazione o, in modo globale, nei settaggi locali del *Common Language Runtime*. Il modello di sviluppo consigliato è quello *hub* e *spoke*: le risorse devono trovarsi in percorsi ben precisi per consentire al CLR la loro individuazione. Ogni lingua deve essere posta in un assembly satellite generato con un comando simile al seguente: *al /t:lib/embed:strings.it.resources/culture:it/out:MioProgetto.resources.dll*. L'argomento è decisamente complesso anche se documentato in maniera particolareggiata nei manuali del Microsoft .NET SDK, non è da escludersi che venga approfondito in un articolo ad hoc.

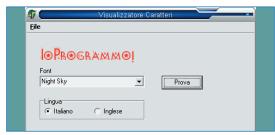



Fig. 8: La lingua può essere cambiata a run-time, anche in questo caso non c'è bisogno di ricompilare

CONCLUSIONI

Come già detto la localizzazione non si conclude con la traduzione, bisogna prestare una notevole cura anche alla gestione dei formati ed operare i cambiamenti in base al feedback da parte degli utenti stranieri. Posso garantirvi che ricevere richieste di traduzione relative ad un software realizzato da voi, da parte di utenti entusiasti, è fonte di grandi soddisfazioni! Vi invito a prendere confidenza con gli strumenti illustrati nell'articolo o, se usate altri linguaggi di programmazione, a cercare librerie e tool di sviluppo con caratteristiche affini. Maggiore è il numero di lingue supportate dall'applicazione più ampia sarà la base dei clienti, la conseguenza immediata è un aumento dei ricavi. Non parliamo mica di bruscolini! Alla prossima, nel frattempo "programmate gente programmate"…

Salvatore Meschini

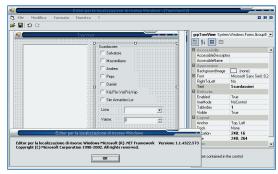



Fig. 9: Lo strumento WinRes.exe è distribuito con il .NET Framework SDK.

Sul web

Delphi GetText v1.1RC1
http://sourceforge.net/projects/dxgettext/

GNU GetText
http://www.gnu.org/software/gettext/gettext.html

poEdit
http://poedit.sourceforge.net

Borland
http://www.borland.com

Microsoft
http://msdn.microsoft.com

Software commerciale
http://www.alchemysoftware.ie

Funzionalità nascoste di Visual Basic.

La funzione SendMessage() in VB

Sistema

Esiste un'API di Windows denominata SendMessage() che consente di realizzare spesso cose che il linguaggio stesso non permette di fare...

Visual Basic è senza dubbio un linguaggio molto potente e rappresenta spesso la via più breve per realizzare progetti che in altri linguaggi potrebbero risultare più complicati. Tuttavia, chiunque abbia esperienza con esso, si sarà certamente accorto che spesso è necessario ricorrere a qualche strumento in più per poter ottenere maggiore flessibilità e, allo stesso tempo facilità nel compimento di molte operazioni. Ognuno di voi avrà subito intuito che il supporto del quale stiamo parlando è rappresentato dalle centinaia di funzioni appartenenti alle API di Windows. Ovviamente, se abbiniamo gli strumenti già insiti nel linguaggio con la possibilità di appoggiarsi alle API disponibili, ci rendiamo subito conto di avere tra le mani uno strumento davvero potente. Una disquisizione completa su tutte le API di Windows meriterebbe numerosi articoli poiché sono davvero tante e spesso "poco comprensibili" senza un opportuno background tecnico. Qui ci occuperemo di descriverne una tra le più interessanti e potenti cercando anche di privilegiare l'aspetto pratico a quello teorico.

LA FUNZIONE SENDMESSAGE()

Sappiamo tutti che Windows è sostanzialmente un sistema operativo basato su finestre e messaggi. Vale a dire che tutti gli oggetti che compongono il sistema operativo stesso, affinché possano effettuare determinate azioni o rispondere a precisi comandi dell'utente, necessitano di particolari avvisi a fronte dei quali compiere determinate operazioni. Come molti di voi sapranno già, ogni "finestra" di Windows è identificabile attraverso il suo *handle* ossia un codice numerico che consente di riferirsi a ciascuna di esse ed a nessun altra. In Visual Basic quest'informazione la ricaviamo attraverso la proprietà *hwnd* di ogni controllo che la espone e, come già spiegato, rappresenta un dato "insito" nel controllo/form stesso. Quest'affermazione è molto importante poiché l'handle rappresenta la chiave principale con la quale la maggior parte delle API di Windows (ivi compresa quella che stiamo per trattare) ha la possibilità di riferirsi ad una o più finestre del sistema operativo senza per questo "ledere" il comportamento delle altre. In particolare, a proposito di questa "classe" di funzioni, è particolarmente interessante quella denominata *SendMessage()*, che consente d'inviare precisi comandi ad una finestra di Windows, consentendo non solo di notificarle eventi, ma soprattutto di ricevere informazioni o altro. La sintassi della funzione *SendMessage()* è la seguente:

```
Private Declare Function SendMessage Lib "user32"
  Alias "SendMessageA" (ByVal hWnd As Long, ByVal wMsg
  As Long, ByVal wParam As Long, lParam As Any) As Long
```

in cui:

- **hWnd:** rappresenta l'handle della finestra alla quale indirizzare il messaggio. Fate attenzione che con il termine *"finestra"* non si vuole affatto "limitare" l'espressione alle sole finestre classiche, ma esso va inteso anche per i controlli come i *Command Button*, le *Textbox*, le *Listbox*, ecc. che in sostanza appartengono proprio ad una classe particolare di finestre.
- **wMsg:** identifica il vero e proprio messaggio da inviare ed è rappresentato, in sostanza, da una costante simbolica che ne facilita l'interpretazione. Vari esempi di questi parametri saranno visti in seguito, quando mostreremo l'uso della funzione.
- **wParam:** questo parametro rappresenta un dato aggiuntivo che può essere inviato alla finestra di destinazione e, naturalmente, può assumere un significato diverso a seconda del tipo di messaggio specificato con *wMsg*.
- **lParam:** quest'ultimo parametro, a differenza dei precedenti, è molto particolare poiché è l'unico ad essere dichiarato *As Any*. Questa peculiarità è molto importante poiché è possibile utilizzare *lParam* per passare alla funzione *SendMessage()* parametri di qualunque tipo, a secondo, ovviamente, del tipo di "azione" che

Articolo completo

Lo stesso articolo è disponibile sul Web:

www.ioprogrammo.it

con inediti paragrafi aggiuntivi.

Sistema

La funzione *SendMessage()* in VB

si sta considerando.

Un banale esempio che mostra l'utilizzo di *SendMessage()* è mostrato di seguito:

```
Label1.Caption = SendMessage(List1.hWnd,
                             LB_GETCOUNT, 0, 0&)
```

La chiamata alla funzione *SendMessage()* mostrata sopra non fa altro che "interrogare" il controllo *List1* ottenendo come risultato il numero di elementi in esso presenti. Come si può vedere, i parametri *wParam* ed *lParam* non sono valorizzati, ma questo è un caso eccezionale. Si noti però il particolare riguardante l'ultimo parametro, specificato come *0&*. Questo dettaglio si rende spesso necessario poiché, essendo *lParam* dichiarato come *As Any,* è necessario specificarne il tipo in maniera precisa, onde evitare problemi ed errori inaspettati.

Variabile Flags

Di seguito sono mostrati i possibili valori che può assumere la variabile Flags relativamente ad un messaggio di tipo *CB_DIR*

DDL_ARCHIVE
File con il flag ARCHIVE impostato

DDL_DIRECTORY
Include le subdirectory. Ognuna d'esse sarà racchiusa tra parentesi quadre.

DDL_DRIVES
Drive letter mostrata nella forma [-x-].

DDL_EXCLUSIVE
Include solo i file con gli attributi specificati. Per default, I file di tipo *R/W* sono inclusi anche se il flag *DDL_READWRITE* non è specificato.

DDL_HIDDEN
File nascosti

DDL_READONLY
File a sola lettura

DDL_READWRITE
File a lettura scrittura senza ulteriori attributi

DDL_SYSTEM
File di sistema.

IL PROGETTO IN VISUAL BASIC

I messaggi che possono essere inviati ad una finestra sono tantissimi e non è semplice elencarli tutti per ogni controllo/finestra di Windows. All'interno del progetto realizzato sono stati inseriti gli esempi che potrebbero tornare più utili, ma anche e soprattutto quelli che consentono di prendere dimestichezza con questa funzione. Per chiunque volesse approfondire questo argomento, consiglio di consultare l'MSDN che offre una descrizione piuttosto esaustiva su tutti i messaggi possibili, suddivisi ovviamente per categoria. Il progetto in Visual Basic realizzato è molto semplice. Esso è composto da una sola form all'interno della quale sono state dichiarate tutte le funzioni, le variabili e le costanti utili a questa dimostrazione. Oltre alla funzione *SendMessage()*, sono state dichiarate altre funzioni che si rendevano necessarie per il corretto funzionamento di alcuni esempi. Esse verranno spiegate più avanti durante la descrizione di ciascun esempio. Il form principale è suddiviso in due parti principali. Nella parte superiore è collocato un controllo *SSTab* suddiviso in quattro sezioni:

- **LISTBOX:** qui sono stati inseriti alcuni esempi che mostrano l'uso della *SendMessage()* applicata alle Listbox.
- **COMBOBOX:** in questa sezione, analogamente alla precedente, sono stati inseriti esempi sull'uso della funzione applicata alle sole Combobox.
- **TEXTBOX:** qui troviamo alcuni esempi sull'uso di *SendMessage()* applicata alle Textbox.
- **MISCELLANEA:** in quest'ultima sezione, infine, sono stati inseriti alcuni esempi che non ricadevano nelle precedenti tre categorie, ma che potevano ugualmente essere interessanti.

Ogni esempio è contrassegnato con un numero progressivo (per sezione) tramite il quale è possibile ottenere una breve descrizione dello stesso (nella parte inferiore della form) semplicemente facendo click con il mouse su di esso. Detto questo, non ci resta che iniziare con il primo esempio, non incluso all'interno delle quattro sezioni precedenti, ma decisamente interessante. Se osserviamo attentamente la form principale, ci accorgeremo della presenza di un'immagine raffigurante una mano nera, proprio nell'angolo superiore sinistro. Se provassimo a trascinare il mouse su di essa, il risultato ottenuto sarebbe quello di trascinamento dell'intera form. Sembrerà strano, ma le righe che consentono di ottenere questo semplice, ed allo stesso tempo sorprendente, risultato sono semplicemente due:

```
Call ReleaseCapture
SendMessage Me.hWnd, WM_NCLBUTTONDOWN,
                              HTCAPTION, 0&
```

Esse vanno inserite all'interno dell'evento *MouseDown* del controllo sul quale si desidera agire e, praticamente, potevano essere inserite anche all'interno dell'evento stesso relativo alla form. La funzione *SendMessage()* utilizzata nell'esempio, peraltro, è applicata come già detto proprio alla form (vedi *Me.hWnd* come primo parametro), ma nulla ci vieta di applicarla ad un controllo qualunque presente su di essa (a patto, ovviamente, di conoscere l'handle a cui fare riferimento e di sostituirlo al posto di *Me.Hwnd*). Prima della chiamata alla funzione oggetto del presente articolo, è bene sottolineare l'importanza di un'altra funzione: *ReleaseCapture()*. La funzione *ReleaseCapture()* è dichiarata come segue:

```
Private Declare Function ReleaseCapture Lib "user32" ()
                                              As Long
```

L'utilizzo di questa funzione prima della chiamata alla *SendMessage()* è fondamentale per un motivo molto semplice. Innanzitutto occorre sapere che il suo scopo è semplicemente quello di rilasciare il cursore del mouse. Il motivo di questo accorgimento è semplice. Quando un utente preme il tasto sinistro del mouse sopra un qualunque controllo della nostra form, quell'oggetto diventa automaticamente l'oggetto attivo. Questo "evento" fa sì che tutti i messaggi inviati (dal semplice movimento del mouse al click di un pulsante, al rilascio dello stesso, ecc.) siano diretti soltanto a quell'oggetto e gestiti da esso. La funzione *ReleaseCapture()* rilascia quindi l'oggetto selezionato in modo da consentire la ridirezione dei messaggi

nella maniera più idonea. Ovviamente qualcuno potrebbe anche chiedersi come mai questo stato di "attivo" non viene ripristinato successivamente dopo la seconda riga di codice. La risposta è semplicissima: perché di questo si occuperanno i tantissimi altri eventi che occorrono all'interno del sistema. Una volta avviata *ReleaseCapture()*, la chiamata alla funzione *SendMessage()*, strutturata nel modo visto sopra, consente dunque di notificare lo spostamento del mouse alla form principale, ingannandola allo stesso tempo e facendole credere, attraverso i parametri passati, che l'operazione di *MouseDown* è avvenuta sulla sua barra del titolo.

LA SEZIONE LISTBOX

Dopo questo piccolo esempio, che credo abbia dimostrato, anche solo in parte, la potenza di una funzione siffatta, possiamo passare a descrivere la prima sezione della nostra form, denominata *LISTBOX*. Essa mostra una serie di esempi di applicazione della funzione *SendMessage()* "ristretta" ai soli controlli Listbox. Al suo interno sono presenti sei diverse applicazioni, ciascuna delle quali consente di effettuare determinate operazioni con questo genere di controlli o, semplicemente, di ottenere informazioni. Il primo esempio, applicato al controllo *List1*, consente di evidenziare un elemento del controllo attraverso il solo spostamento

Fig. 1: **La sezione LISTBOX.**

del cursore del mouse su di esso.
Il codice che consente di ottenere questo risultato è il seguente:

```
Private Sub List1_MouseMove(Button As Integer, Shift
                    As Integer, X As Single, Y As Single)
  Dim Item&, PosX&, PosY&, Param&
  ' Converti la posizione X ed Y in Pixel per passare
  ' correttamente i parametri a SendMessage(),
  ' espressi, per la listbox, in twips 1) Metodo
    'PosX = List1.Parent.ScaleX(X,
                         List1.Parent.ScaleMode, vbPixels)
    'PosY = List1.Parent.ScaleY(Y,
                         List1.Parent.ScaleMode, vbPixels)
    'Param = CLng(PosX) + &H10000 * CLng(PosY)
  ' 2) Metodo
   Param = (CInt(Y / Screen.TwipsPerPixelY) * 2 ^ 16)
                         + CInt(X / Screen.TwipsPerPixelX)
Item = SendMessage(List1.hWnd, LB_ITEMFROMPOINT,
                                       0, ByVal Param)
  If Item < List1.ListCount Then
     List1.ListIndex = Item
    List1.ToolTipText=List1.List(Item)
 End If

End Sub
```

Il messaggio che consente di ottenere questo risultato è rappresentato dalla costante *LB_ITEMFROMPOINT*. Esso non fa altro che sfruttare le coordinate *X* ed *Y* del puntatore del mouse per "ritornare" come risultato il corretto numero di riga. La prima particolarità di quest'applicazione è rappresentata dal fatto che i parametri passati attraverso la *SendMessage()* alla Listbox devono essere espressi necessariamente in pixel. Si noti a questo proposito la conversione delle coordinate in quest'unità di misura, operazione questa che si rende necessaria proprio perché l'unità di misura utilizzata per le Listbox è invece il *twip*. Si osservi inoltre che *wParam* non è utilizzato e che *lParam* esprime le coordinate del mouse secondo la seguente "convenzione":

- La *low word* rappresenta la coordinata *X* di un punto, relativamente all'angolo superiore sinistro, della client area della Listbox.
- L'*high word* rappresenta la coordinata *Y* di un punto, relativamente all'angolo superiore sinistro, della client area della Listbox.

Un ultimo dettaglio importante è rappresentato anche dal fatto che l'evento *Click* della Listbox, attraverso la selezione di un item ottenuta con il codice precedente, non è affatto scatenato. Il secondo esempio racchiude in sé due dimostrazioni sull'uso della *SendMessage()* applicata a questo tipo di controlli. La prima è forse più banale della precedente poiché, attraverso tre semplici chiamate alla funzione *SendMessage()*, consente di ottenere alcune informazioni su *List2* nel momento in cui è avviato il programma oppure quando si seleziona una qualunque riga. Le informazioni mostrate (esempio 4) sono: numero di righe totali, riga evidenziata e lunghezza dell'item correntemente selezionato. Considerata la semplicità d'interpretazione di quest'applicazione, credo non occorra dire altro in proposito. La seconda dimostrazione, invece, è decisamente più interessante poiché consente di ricercare e selezionare automaticamente un elemento di *List2* semplicemente digitando una stringa all'interno di una Textbox. Il codice che permette di realizzare ciò è il seguente:

Sistema

La funzione *SendMessage()* in VB

Messaggi

I messaggi che possono essere inviati ad una finestra sono tantissimi e non è semplice elencarli tutti per ogni controllo/finestra di Windows. All'interno del progetto realizzato sono stati inseriti gli esempi che potrebbero tornare più utili, ma anche e soprattutto quelli che consentono di prendere dimestichezza con questa funzione.

ReleaseCapture()

La funzione *ReleaseCapture()* rilascia l'oggetto selezionato in modo da consentire la ridirezione dei messaggi nella maniera più idonea.

Sistema

La funzione *SendMessage()* in VB

```
Private Sub Text1_Change()
   Dim Item&
   If Option1 Then
      Item = SendMessage(List2.hWnd,
                    LB_FINDSTRINGEXACT, -1, Text1.Text)
   Else
      Item = SendMessage(List2.hWnd,
                         LB_FINDSTRING, -1, Text1.Text)
   End If
   List2.ListIndex = Item
   If List2.ListIndex > -1 Then List2.TopIndex = Item
End Sub
```

Il controllo Text1 rappresenta la casella di testo in cui inserire il valore da ricercare. Come si sarà potuto notare, l'esempio effettua due diverse chiamate alla funzione *SendMessage()* che si differenziano soltanto per il valore del secondo parametro. Esso può essere:

- **LB_FINDSTRINGEXACT:** se viene sfruttato questo messaggio, la riga della listbox verrà evidenziata solo quando verrà inserita, all'interno di *Text1*, una stringa esattamente identica ad uno degli item di *List2*.
- **LB_FINDSTRING:** analogamente a quanto già detto, questo messaggio consente di selezionare una qualunque riga di *List2* mano a mano che viene digitata una stringa in *Text1*.

Il penultimo esempio consente di copiare il contenuto di una listbox in un'altra in maniera molto più rapida di quella che sfrutta le proprietà ed i metodi del controllo stesso:

```
Private Sub Image12_Click()
   Dim Index As Long
   Dim ItmData As Long
   Dim Items As Long
   Dim ItmText As String * 255
   ' Blocca il redraw della List1
     LockWindowUpdate List1.hWnd
   ' Reset della listbox di destinazione
     SendMessage List1.hWnd, LB_RESETCONTENT, 0,
                                           ByVal 0&
   ' Numero di elementi all'interno di List2
     Items = SendMessage(List2.hWnd, LB_GETCOUNT,
                                       0&, ByVal 0&)
   For Index = 0 To Items - 1
      ' Preleva l'item corrente da List2
        SendMessage List2.hWnd, LB_GETTEXT, Index,
                                       ByVal ItmText$
      ' Aggiungilo alla listbox List1
        SendMessage List1.hWnd, LB_ADDSTRING, 0&,
                                       ByVal ItmText$
      ' ... procedi in maniera analoga a prima per ItemData
        ItmData = SendMessage(List2.hWnd,
                        LB_GETITEMDATA, Index, ByVal 0&)
        SendMessage List1.hWnd, LB_SETITEMDATA,
                                     Index, ByVal ItmData
   Next
   ' Sblocca il redrawing
     LockWindowUpdate 0
   ' Imposta la scrollbar orrizontale
      SendMessage List1.hWnd,
           LB_SETHORIZONTALEXTENT, 130, ByVal 0&
End Sub
```

Anche questo esempio è stato ben commentato per facilitarne la comprensione e credo non occorra molto in proposito. L'ultima chiamata alla funzione *SendMessage()*, in particolare, è stata inserita solo per consentire di leggere item più lunghi di quanto non poteva essere mostrato attraverso le dimensioni "ridotte" della client area della listbox. Ai fini dell'operazione di copia è pertanto ininfluente. L'ultimo esempio, infine, mostra come sia possibile eliminare e, successivamente, inserire un item all'interno di questi controlli. Il codice inserito all'interno del progetto è il seguente:

```
Private Sub Image17_Click()
  Dim NewElem As String
  ' Elimina il primo elemento della lista
    SendMessage List2.hWnd, LB_DELETESTRING, 0, 0&
    List2.ListIndex = 0
  ' Inserisci un nuovo elemento
    NewElem = "Francesco Lippo"
    SendMessage List2.hWnd, LB_INSERTSTRING, 0,
                                           NewElem$
End Sub
```

Il messaggio *LB_DELETESTRING* sfrutta solo *wParam* come argomento e, com'è facile intuire, esso rappresenta l'item da eliminare. Analogamente, *LB_INSERTSTRING* consente l'inserimento di un nuovo elemento (nell'esempio rappresentato da *NewElem*) nella posizione specificata da *wParam*. Il terzo parametro, passato attraverso la funzione, è nel primo caso l'item da eliminare e, nel secondo caso, la posizione ove inserire la nuova stringa.

CONCLUSIONI

Questa brevissima panoramica sulla funzione *SendMessage()* spero avrà suscitato l'interesse della maggior parte di voi. Quello che deve essere chiaro è senza dubbio il fatto che ogni controllo/form di Visual Basic espone proprietà e metodi che non sono "novità" del linguaggio stesso, ma comode "visualizzazioni" di proprietà e metodi già precostituiti all'interno del sistema operativo. Prima di concludere vorrei solo sottolineare che il sistema operativo utilizzato per la costruzione degli esempi è stato Windows XP e che la versione di Visual Basic utilizzata è la 6.0 Professional.

Francesco Lippo

ComboBox

L'intera sezione dedicata alle Combobox è composta da soli quattro esempi. Ovviamente le possibilità di applicazione offerte per questo tipo di controllo, così come per il precedente, sono molte di più.

Le novità per gli sviluppatori.

Windows Mobile 2003

Pocket PC

Lo scorso mese di Giugno Microsoft ha presentato Windows Mobile 2003 per Pocket PC, l'ultima versione del software che permette all'utenza professionale mobile di connettersi con persone e accedere a informazioni e dati ovunque e in qualsiasi momento. Le novità per gli sviluppatori sono numerose e interessantissime: un nuovo sistema operativo, nuovi tool di sviluppo e nuove API per il supporto nativo delle connessioni Wi-Fi e Bluetooth.

Pocket PC 2003 è basato su Windows CE .NET 4.2, il nuovo sistema operativo capace di rendere i dispositivi palmari più veloci e affidabili. Pocket Internet Explorer, l'emulatore, i servizi di rete e Windows Media player sono alcune delle aree che sono state largamente migliorate nella nuova versione di Windows CE. Per esempio il nuovo Pocket Internet Explorer supporta l'HTML 4.01, i CSS, l'XHTML e il WML 2.0. Inoltre, l'integrazione nella ROM del .NET Compact Framework consente di migliorare notevolmente le prestazioni e la robustezza dei software realizzati con la tecnologia .NET. Per la realizzazione di applicazioni per Windows Mobile 2003 è possibile utilizzare sia Visual Studio .NET 2003 per lo sviluppo di Codice "managed" sia il nuovo eMbedded Visual C++ 4.0 con Service Pack 2 per lo sviluppo di Codice "unmanaged". Non è possibile, invece, usare l'eMbedded Visual Basic che non è più supportato dal nuovo SDK.

EMBEDDED VISUAL C++

Oltre alla possibilità di utilizzo delle nuove API messe a disposizione da Windows CE.NET 4.2, le novità dell'eMbedded Visual C++ 4.0 riguardano la fase di debugging e la presenza di nuovi remote tool. La novità più interessante è il supporto in C++ della gestione delle eccezzioni con la struttura *try-catch-finally*, che garantisce la scrittura di codice più portabile e più flessibile. Un'altra interessante caratteristica è la possibilità di connettersi ed eseguire il debug di un processo attivo sul dispositivo. Se un'applicazione è bloccata senza che il debugger sia attivo, è possibile connettersi al processo selezionando dal menu *Build* la voce *"Start Debugging"* e poi *"Attach to Windows CE Process"*. Dopo aver selezionato il processo tra quelli attivi sul dispositivo e indicato il percorso del file eseguibile sul PC, il debugger si connette al processo selezionato aiutando ad individuare le ragioni del blocco. Altre due interessanti novità sono la possibbilità di catturare eccezioni non gestite senza dover terminare l'applicazione e la possibilità di bloccare tutti i thread di un'applicazione utilizzando un unico breakpoint. Sono presenti, inoltre, due nuovi tool: il *Remote Call Profiler* e il *Remote Performance Monitor.* Il primo consente allo sviluppatore di tracciare le chiamate di un'applicazione in esecuzione, mentre il secondo gli permette di monitorare alcune metriche predefinite in real time. Altre importanti novità, di cui gli sviluppatori saranno particolarmente contenti, sono il supporto per gli *"intrinsics"* (le funzioni sono compilate inline piuttosto che come chiamata di funzione), un wizard per la creazione di componenti ATL out-of-process (per creare server COM out-of-process), e la possibilià di usare le *Standard Template Library* (STL, che forniscono l'accesso ad un insieme di algoritmi e strutture dati largamente diffuse).

EMBEDDED VISUAL BASIC

L'SDK del Pocket PC 2003 non supporta lo sviluppo in eMbedded Visual Basic e i nuovi dispositivi non includono nella ROM nè l'eMbedded Visual Basic runtime nè le DLL di ADOCE. È comuncque possibile scaricare entrambi questi software da Internet ed installarli nella RAM per continuare ad usare il Pocket PC 2002 SDK per sviluppare applicazione eVB per Pocket PC 2003. Gli sviluppatori, invece, che desiderano aggiornarsi possono migrare verso Visual Basic .NET per usufruire dei vantaggi dei nuovi tool. Dal punto di vista del linguaggio la novità più significativa è il fatto che Visual Basic .NET è un moderno linguaggio Object Oriented. Inoltre VB.NET utilizza le potenti librerie del .NET Compact Framework e supporta in maniera nativa sia gli XML Web Service sia la gestione strutturata delle ec-

Convivenza

È possibile installare contemporaneamente su uno stesso PC i vecchi eMbedded Visual Tools 3.0, l'eMbedded Visual C++ 4.0 con SP 2 e il Visual Studio .NET senza rischiare conflitti di alcun tipo.

Pocket PC

Windows
Mobile 2003

cezioni. Poichè non c'è nessun wizard o processo automatico per convertire i progetti sviluppati in eMbedded Visual Basic in Visual Basic .NET, lo sviluppatore deve modificare manualmente il Codice tenendo in considerazione i seguenti punti:

- L'*eMbedded Visual Basic* è un linguaggio simile a VB Script per cui ci sono differenze significative nella gramatica del linguaggio.
- Le librerie comuni dell'*eMbedded Visual Basic* devono essere ricompilate per il *.NET Compact Framework*. Inoltre, l'implementazione e l'uso delle librerie sarà differente poichè VB.NET supporta l'uso delle classi
- La navigazione dell'applicazione è implementata differentemente dall'*eMbedded Visual Basic*.
- L'accesso ai dati è gestito attraverso un sottoinsieme di *ADO.NET*. Il *.NET Compact Framework* fornisce un *managed data provider* per SQL Server CE 2.0, ma non include classi managed per l'accesso ai dati locali del datastore come CEDB o Pocket Access, che sono invece comunemente usati nelle applicazioni *eMbedded Visual Basic*.
- L'*eMbedded Visual Basic* fornisce, solo attraverso software di terze parti, la chiamata a componenti remoti.VB.NET invece supporta in maniera nativa i Web Services che sono considerati il meccanismo fondamentale per l'integrazione dei dati. È molto probabile, dunque, la necessità di dover riscrivere tutto la parte di codice che riguarda l'integrazione con altri sistemi.
- La gestione delle eccezioni in *eMbedded Visual Basic* consiste nell'utillzzo delle strutture *"On Error Resume Next"* e *"If Err.Number <> 0 Then"*. VB.NET invece utilizza il blocco *"Try-Catch-Finally"* per migliorare la robustezza e la tolleranza ai guasti del codice.

Per comprendere meglio la differenza nella gestione delle eccezioni in *eMbedded Visual Basic* rispetto a Visual Basic .NET, vediamo in dettaglio un esempio:

Codice A

```
'Apri il file
'Variabili
Dim sLine As String
Dim file As FILECTL.file
Set file = CreateObject("FILECTL.file")
'Gestione dell'errore utilizzando "resuming next"
On Error Resume Next
'Apri il file
file.Open "\appdata.txt", fsModeInput
'Controlla la presenza di un errore
If Err.Number <> 0 Then
    MsgBox "Il file non può essere aperto!", vbCritical,
                                                "Errore"
    Exit Sub
End If
'Loop through file
Do While Not file.EOF
    'Leggi una linea alla volta
    sLine = file.LineInputString
    'Controlla la presenza di un errore
    If Err.Number <> 0 Then
        MsgBox "I dati non possono essere letti!",
                                    vbCritical, "Errore"
        Exit Sub
    End If
Loop
'Chiudi il file
file.Close
```

Il codice presentato apre un file di testo e legge una linea per volta fino al raggiungimento dell'*end of file* (EOF). In *eMbedded Visual Basic* è necessario aggiungere un *"If Err.Number <>0"* per catturare e gestire ogni eccezione. La linea di codice inziale *"On Error Resume Next statement"* indica che in caso di errore il programma continua con l'esecuzione della linea successive. L'oggetto *Err* contiene informazioni su ogni possible errore che si è verificato. Controllando dopo ogni riga di codice se la proprietà *Number* dell'oggetto *Err* è diversa dal valore di default *(0)*, lo sviluppatore puo' verificare e gestire la presenza di un errore, ma non sa in quale riga esattamente esso si è verificato. Vediamo come può essere riscritto lo stesso codice in VB.NET.

Codice B

```
'Apri File
'Variabili
Dim sLine As String
Dim file As StreamReader
'Gestione dell'errore con Try-Catch-Finally
Try
    file = New System.IO.StreamReader("\appdata.txt")
    While file.ReadLine <> Nothing
        sLine = file.ReadLine
    End While
    file.Close()
Catch ex As Exception
    Select Case ex.Message
        Case "FileNotFoundException"
            MsgBox("Il file non può essere aperto!",
                            MsgBoxStyle.Critical, "Error")
        Case Else
            MsgBox("I dati non possono essere letti!",
                            MsgBoxStyle.Critical, "Error")
            file.Close()
    End Select
Finally
    Cursor.Current = Cursors.Default
End Try
```

Tutto il codice che gestisce la gestione del file è posto nel blocco *Try*, il codice che riguarda ogni possible errore nel blocco *Catch* e il codice che deve essere eseguito comunque dopo il *Try-Catch* nel blocco *Finally*.

Andrea Aiello

Migrazioni

Nel migrare un'applicazione da eMbedded Visual C++ 3.0 a eMbedded Visual C++ 4.0 non è possibile aggiornare automaticamente i file di progetto. Il modo migliore per risolvere questo problema è di creare dei file di progetto vuoti nel nuovo ambiente di sviluppo e aggiungere i file sorgenti manualmente.

Robustezza

Il nuovo emulatore è molto più robusto e supporta 3 modalità: Pocket PC 2003, Pocket PC Phone Edition 2003 con supporto del modulo GSM esterno WaveCom WMOD2B e Pocket PC Phone Edition 2003 con Virtual Radio. Inoltre supporta il virtual switch, il drive mapping, e le GAPI.

L'evoluzione del mobile browsing con WAP 2.0 e XHTML-MP

XML accende i cellulari

XHTML-MP

Dopo un periodo di empasse, il settore delle comunicazioni mobili è ritornato alla ribalta.

Il biennio 2003-2004 viene indicato dai maggiori istituti di ricerca come il periodo del definitivo decollo delle comunicazioni mobili Internet-oriented. Sotto il profilo strettamente tecnologico, una grossa spinta deriverà, oltre che da una maggiore stabilità e migliori prestazioni delle reti mobili, dall'adozione del protocollo WAP 2.0. A differenza delle versioni precedenti, il WAP 2.0 prevede la piena interoperabilità con i protocolli di Internet (HTTP 1.1 e TCP/IP). Anche il linguaggio di markup si è notevolmente evoluto: dal povero e monocromatico WML delle prime versioni si passa all'XHTML-MP, appartenente alla famiglia di linguaggi XHTML che, secondo i progetti del W3C, è destinata a sostituire definitivamente il glorioso HTML. In quest'articolo, faremo una panoramica sull'XHMTL-MP, mostrando le caratteristiche del linguaggio attraverso una serie di esempi. Per testare quest'ultimi, ci avvarremo di uno degli ambienti integrati più diffusi per lo sviluppo di applicazioni destinate a terminali mobili, il Nokia Mobile Internet Toolkit (NMIT), disponibile per il download all'indirizzo www.forum.nokia.com. Nella sua installazione base, il toolkit contiene un simulatore di terminale mobile per PC per la preview dei contenuti. Si tratta del Nokia Mobile Browser (NMB) e rappresenta l'implementazione di riferimento del mobile Internet browser che sarà utilizzato nei terminali Nokia di nuova generazione, compatibili con la versione 2.0 del protocollo WAP.

XHTML: UN HTML RIVISITATO IN CHIAVE XML

Gennaio 2000: il W3C (World Wide Web Consortium, www.w3.org) pubblica la versione ufficiale delle specifiche di XHTML 1.0 (eXtensible HTML), un linguaggio di markup che riformula gli elementi base dell'HTML 4 alla luce delle regole sintattiche dell'XML 1.0. Il motivo di quest'opera di ridefinizione dell'HTML (non sono previsti infatti nuovi tag …) fu la volontà del W3C di ritornare ad un linguaggio che definisse solo il contenuto di un documento, senza preoccuparsi degli aspetti legati alla presentazione. Uno dei principali difetti delle ultime versioni dell'inossidabile HTML, infatti, era quello di mescolare contenuto e stile con il risultato di un codice difficile da gestire e poco portabile (cross-browsing). Dopo aver definito la versione 1.0 di XHTML, il W3C fu impegnato nella strutturazione del linguaggio in moduli. La modularizzazione dell'XHMTL consentì di definire sottoinsiemi del linguaggio adatti alle caratteristiche di client Web diversi dai tradizionali PC. Sotto la spinta dell'OMA (Open Mobile Alliance, *www.openmobilealliance.org*), infatti, nacque l'XHTML Basic, un subset di XHTML specificamente pensato per dispositivi mobili dalle limitate risorse hardware e software. L'XHTML Basic venne poi esteso dall'OMA con il supporto dei WAP CSS, una versione "mobile" della specifica CSS Level 2, dando così origine all'XHTML-MP (Mobile-Profile).

STRUTTURA DI UN DOCUMENTO XHTML-MP

Analizziamo la struttura base di un documento XHMTL-MP:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML
                                    Mobile 1.0//EN"
"http://www.wapforum.org/DTD/xhtml-mobile10.dtd" >
<html xmlns = "http://www.w3.org/1999/xhtml">
<head> Sezione HEAD </head>
<title> Sezione TITLE </title>
<body> Sezione BODY </body>
</html>
```

Tutti i documenti XHTML iniziano con un prologo costituito da due righe particolari. La prima indica la versione di XML e (opzionalmente) il set di caratteri utilizzato nel documento, mentre la seconda è costituita dalla direttiva *DOCTYPE* che specifica la *DTD (Document Type Declaration)* a cui il documento è conforme (nella fattispecie quella dell'XHTML-MP 1.0). Subito dopo il prologo, troviamo l'elemento *<html>*, che racchiude il documento XHTML-MP vero e proprio. L'attributo *"xmlns"* specifica il namespace predefinito per XHTML. Un documento XHTML è strutturato in un

Documenti XML well-formed e validi

Secondo la terminologia XML, un documento è well-formed (ben formato) se rispetta le regole sintattiche dell'XML per la corretta specifica degli elementi, degli attributi e delle entità. Queste regole prevedono, ad esempio, il bilanciamento dei marcatori (ad ogni tag di apertura deve corrisponderne uno di chiusura) o l'uso delle virgolette per racchiudere i valori degli attributi. Un documento è invece valido se, innanzitutto, è *well-formed* e se rispetta le regole di nidificazione definite nella *DTD (Document Type Definition)* che viene specificata all'inizio del documento stesso.

XHTML-MP

XML
accende i cellulari

head (intestazione) e in un body (corpo) delimitati dai corrispondenti tag. L'head deve contenere un titolo *(tag <title>)*, che comparirà nella parte superiore del display del cellulare. Il body racchiude, invece, i contenuti: testo strutturato in paragrafi, elenchi, link, immagini, tabelle, form, menù…

PRIMI PASSI CON XHTML-MP

Per iniziare a prendere familiarità con il linguaggio, consideriamo un semplice documento XHTML-MP costituito da un paragrafo contenente testo. Per la formattazione del testo, esistono diverse possibilità tra le quali quella di definire abbreviazioni *(<abbr>)*, acronimi *(<acronym>)*, indirizzi *(<address>)*, citazioni *(<cite>* e *<blockquote>)*, frammenti di codice di un linguaggio di programmazione *(<code>* e *<var>)*, definizioni *(<dfn>)*, testo in corsivo *(<em>)* o grassetto *(<strong>)*, ecc.

Fig. 1: **Paragrafo contenente testo formattato.**
Fig. 2: **Lista di definizioni.**

```
<body>
  <p>
    Testo Normale <br/>
    <em> Testo Corsivo </em> <br/>
    <strong> Testo Grassetto </strong> <br/>
    <q> Testo Quotato </q> <br/>
<abbr> Abbreviazione </abbr> <br/>
<acronym> Acronimo </acronym> <br/>
<address> Indirizzo </address> <br/>
<cite> Citazione </cite> <br/>
<code> Codice </code> <br/>
<dfn> Definizione </dfn> <br/>
<var> Variabile </var> <br/>
   </p>
  </body>
```

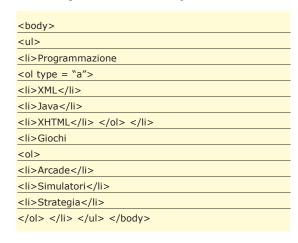
L'effetto che si ottiene sul Nokia Mobile Browser dell'NMIT è raffigurato in Fig. 1. Come in HTML, esiste poi la possibilità di strutturare i contenuti in sezioni e sottosezioni con i tag *<h1>, …, <h6>*.

LISTE ED ELENCHI

In XHTML-MP esistono tre tipi di liste: liste di definizioni, elenchi numerati *(ordered)* e elenchi puntati *(unordered)*. Una lista di definizioni *(<dl>)* comprende un elenco di termini *(<dt>)* ciascuno con la relativa definizione *(<dd>)*. In Fig. 2, viene riportato l'esempio di una lista di definizioni relativa al seguente frammento di codice:

```
<body>
  <h2> Definition list </h2>
<dl> <dt> WAP </dt>
  <dd> Wireless Application Protocol </dd>
<dt> CSS </dt>
  <dd> Cascading Style Sheet </dd>
<dt> XHTML-MP </dt>
  <dd> XHTML Mobile Profile </dd>
</dl>
</body>
```

L'esempio successivo mostra, invece, l'uso combinato di elenchi puntati e numerati (Fig. 3).

```
<body>
<ul>
<li>Programmazione
<ol type = "a">
<li>XML</li>
<li>Java</li>
<li>XHTML</li> </ol> </li>
<li>Giochi
<ol>
<li>Arcade</li>
<li>Simulatori</li>
<li>Strategia</li>
</ol> </li> </ul> </body>
```

L'elemento <ul> definisce un elenco puntato, le cui voci (*"Programmazione"* e *"Giochi"*, nel nostro esempio) sono specificate dall'elemento *<li>*. All'interno di ciascuna di queste voci, è nidificato un elenco numerato (elemento *<ol>*).

Form - Dalla programmazione HTML, è noto che i form sono componenti mediante i quali un utente può inviare informazioni (dati di login, di registrazione, e-mail, commenti, risposte a sondaggi, ecc..) all'applicazione residente lato server. I dati contenuti nel form vengono spediti, mediante un bottone "Submit", ad

Fig. 3: **Elenchi puntati e numerati.**
Fig. 4: **Textbox e TextArea**

Nokia Mobile Internet Toolkit

Il *Nokia Mobile Internet Toolkit* (NMIT) è un ambiente integrato progettato per supportare lo sviluppatore nell'authoring di contenuti secondo i principali standard che si stanno affermando nel settore dell'Internet Mobile:

***WML 1.x, WMLScript, WBMP* per applicazioni *WAP 1.x*;**
XHTML-MP* e *WAP CSS* per applicazioni *WAP 2.0;
***WAP Push* (*SI, SL, CO, Multipart*) per servizi di alerting;**
***MMS* e *SMIL* per la messaggistica multimediale.**

La versione 3.1 gira sotto WIN XP, WIN 2000 *(SP2)*, WIN NT 4.0 *(SP6)*, WIN 98 e necessita di JRE 1.3 (raccomandato 1.3.1), mentre la versione 4.0 è adatta per WIN XP o 2000 *(SP 2)* e richiede *JRE 1.4.1.* Entrambe le versioni sono disponibili per il download all'indirizzo www.forum.nokia.com

uno script CGI residente sul server, che si occuperà di processarlo e inviare la pagina di risposta all'utente. E' buona regola affiancare il tasto "Submit" con un tasto "Reset", che permette all'utente di azzerare il contenuto del form. La struttura base di un modulo è pertanto la seguente:

```
<form method="get|post" action="/cgi-bin/nome_script.cgi">
...
<input type = "submit" value = "Invia i dati" />
    <input type = "reset" value = "Azzera" />
</form>
```

Come in HTML, un modulo viene definito utilizzando l'elemento *<form>*, all'interno del quale vengono specificati il metodo (*GET* o *POST*) per inviare i dati contenuti e lo script CGI a cui sono indirizzati. Un form contiene tipicamente una serie di elementi per l'inserimento dei dati da parte dell'utente. Ad esempio, l'elemento *<input>* con attributo type valorizzato a *"text"* consente di creare un box dove inserire testo, tipo dati di login o dati anagrafici (telefono, mail, ecc..). Se i dati da inserire occupano più di una linea (commenti, corpo di una e-mail, ecc..), allora conviene utilizzare una *textarea*.

```
<form method="post" action="/cgi-bin/nome_script.cgi">
    <h2> Feedback </h2>
<label>Commenti:
  <textarea name = "commenti" rows = "3" cols = "15">
        Inserisci i commenti:
  </textarea>
  </label>
  <label>E-mail Address:
  <input name = "email" type = "text" size = "15" />
  </label>
  <p>
  <input type = "submit" value = "Invia" />
  <input type = "reset" value = "Annulla" />
  </p>
</form>
```

Il frammento di codice presentato è relativo ad un form composto da due elementi (Fig. 4):

1) una textarea *"commenti"* composta da 3 righe di 15 caratteri ciascuna;
2) un textbox *"email"* di dimensione pari a 15 caratteri.

Se l'attributo *type* dell'elemento *<input>* assume il valore *"password"* anziché *"text"*, il testo immesso dall'utente appare mascherato.

Menù - I form possono contenere anche uno o più menù, in cui vengono proposte all'utente liste di opzioni predefinite. I menù vengono costruiti usando gli elementi *<select>* ed *<option>* (per le voci del menù). La Fig. 5 mostra due caselle combinate: la prima ha l'attributo multiple valorizzato per cui l'utente può scegliere anche più di una voce tra quelle proposte, mentre nella seconda può selezionarne solo una. La selezione di default (attributo selected) prevede le voci *"A"* e *"C"* per il primo menù e la voce *"A"* per il secondo.

```
<form method="post" action="/cgi-bin/nome_script.cgi">
    <h2> Menù </h2>
    <select name="Menu1" size="2" multiple="multiple">
      <option selected="selected">A</option>
    <option>B</option>
      <option selected="selected">C</option>
      <option>D</option>
  </select>
    <select name="Menu2">
    <option>A</option>
    <option>B</option>
    <option>C</option>
    <option>D</option>
    </select>
<p>
  <input type = "submit" value = "Invia" />
  <input type = "reset" value = "Annulla" />
  </p>
</form>
```

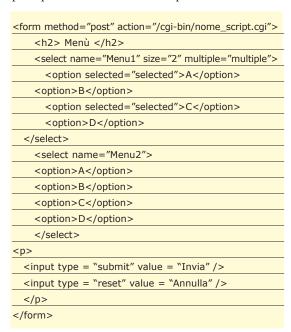



Tabelle - Le tabelle sono strutture nate per contenere dati e disporli in modo da poter essere meglio fruiti dall'utente. In HTML, vengono ormai massicciamente impiegate per definire il layout di una pagina, determinando quel mescolamento di contenuto e presentazione che costituisce il principale difetto dell'HTML in materia di cross-browsing e accessibilità, oltre che di mera purezza stilistica. In XHTML, le tabelle ritornano ad essere tabelle e basta. La struttura tipo di una tabella è mostrata nel seguente frammento di codice:

```
<table>
<tr> <th>PRIMA Colonna</th>
<th>SECONDA Colonna</th> </tr>
<tr> <td>Prima cella della prima riga</td>
<td>Seconda cella della prima riga </td> </tr>
...
</table>
```

Tenendo presente questo schema, è possibile creare tabelle di qualsivoglia complessità. Le celle possono contenere testo, link, immagini, ecc. Per migliorare la frui-

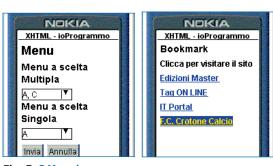



Fig. 5: I Menu'.
Fig. 6: I link.

XHTML-MP

W3C XHTML Validation Tool

All'indirizzo http://validator.w3.org/file-upload.html è disponibile un servizio online di validazione di documenti XHTML e HTML. Il documento da validare deve essere selezionato (pulsante *Sfoglia...*) e quindi uploadato (pulsante *Validate this File*) sui server W3C. È possibile specificare anche il set di caratteri utilizzato nel documento (*Encoding*) e la DTD rispetto alla quale si vuole verificare la conformità (*Type*). In caso contrario, queste ultime due informazioni vengono ricavate automaticamente dal sistema di validazione W3C.

XHTML-MP

zione dei contenuti, può essere utile utilizzare header di colonna *(<th>)* o didascalie *(<caption>)*. Per chi proviene dal mondo HTML, è bene precisare che il tag *<table>* non supporta gli attributi *"align"*, *"border"*, *"cellpadding"* (spazio tra contenuto di una cella e bordo) e *"cellspacing"* (spazio tra le singole celle). Ma questo non dovrebbe meravigliare: in ottica XHTML, allineamenti, bordi, spaziature e colori sono separati dal contenuto e vengono realizzati con i fogli di stile.

LINK E IMMAGINI

Per completare la carrellata sugli elementi dell'XHTML-MP, non ci resta che vedere come inserire link e immagini nelle nostre pagine. Per definire un link ipertestuale viene usato l'elemento *<a>*, che prevede l'attributo *"href"* per specificare l'URL target e l'attributo *"accesskey"* per consentire all'utente di attivare il link anche premendo un tasto del tastierino numerico del telefonino. In Fig. 6, riportiamo la preview del codice seguente che mostra l'utilizzo dei link:

```
<body>
<h3>Bookmark</h3>
<p><strong>Clicca per visitare il sito</strong></p>
<p><a href = http://www.edmaster.it accesskey=
                              "1">Edizioni Master</a></p>
<p><a href = http://www.tagonline.it accesskey="2">
                                     Tag ON LINE</a></p>
<p><a href = http://www.itportal.it accesskey=
                                  "3">IT Portal</a></p>
<p><a href = http://www.fccrotone.it accesskey=
                          "4">F.C. Crotone Calcio</a></p>
</body>
```

Vediamo adesso il codice per inserire un'immagine nella nostra pagina XHTML (Fig. 7):

```
<body> <p>
    <img src = "ioprog.gif" alt = "ioProgrammo"/>
</p> </body>
```

Fig. 7: Uso delle immagini.

W3C Tidy

HTML Tidy è un tool, scritto da David Raggett e liberamente scaricabile all'indirizzo www.w3.org/People/Raggett/tidy per la conversione automatica di documenti HTML in XHTML. Risulta molto utile agli sviluppatori che stanno progettando di rendere accessibile il proprio sito a dispositivi alternativi al PC o che vogliono aderire al nuovo standard XHTML per esigenze di maggiore rigore sintattico e purezza stilistica. È disponibile sia versione in C++ che la versione in Java, entrambe distribuite con licenze a sorgenti liberi.

Il Nokia Mobile Browser supporta il formato GIF per le immagini a colori e WBMP per quelle monocromatiche. Un altro elemento utile quando si programma in XHTML-MP è *<base>* (da inserire all'interno dell'elemento *<head>*) che definisce l'URL di base a partire dal quale vanno considerati tutti gli URL relativi.
Ad esempio, l'istruzione:

```
<base href=http://www.miositowap.com/>
```

consente di scrivere nel body URL relativi del tipo:

```
<a href="/Esempi/App1/index.xhtml"> App1 </a>
```

WAP CSS E CSS-MP 1.0

Una caratteristica peculiare di XHTML-MP è il supporto dei "fogli di stile a cascata" *(Cascading Style Sheets, CSS)*. Come è noto dalla programmazione Web, un file CSS è un insieme di direttive separate dal contenuto di un documento, che stabiliscono come questo debba essere presentato all'interno di un browser (font, colori, posizionamento,…). Il WAP Forum (*www.wapforum.org*) ha definito la specifica WAP CSS, una versione della specifica CSS Level2 appositamente studiata per documenti XHTML-MP. Parallelamente, il W3C è impegnato nell'opera di standardizzazione di CSS-MP 1.0 (*www.w3.org/TR/css-mobile*), che attualmente è a livello di *"Candidate Recommendation"*. Gli user-agent compatibili con WAP CSS sono in grado di interpretare i fogli si stile W3C CSS-MP 1.0. Passiamo ora a toccare con mano la potenza dei fogli si stile nel rendering dei contenuti. Cominciamo col dire che i fogli di stile possono essere di due tipi: esterni o interni. Nel primo caso, il foglio di stile è un file CSS separato dal contenuto XHTML-MP: le coordinate dello stylesheet vengono specificate all'interno dell'elemento *<link>*.
La sintassi è la seguente:

```
<head>
<link href="stili.css" type="text/css" rel="stylesheet"/>
</head>
```

Questa direttiva indica semplicemente di utilizzare il file "stili.css" contenuto nella stessa directory del documento XHTML-MP. E' possibile anche usare fogli di stile "interni" al documento XHTML-MP utilizzando l'elemento *<style>*: in questo caso gli stili si applicano al documento in cui sono definiti. Ad esempio, con il seguente frammento di codice

```
<head>
<title>XHTML - ioProgrammo</title>
<style type="text/css">
ol {color: blue}
ul {list-style-type: circle; color: red}
</style>
</head>
```

specifichiamo il formato e lo stile degli elenchi puntati e numerati. I primi *(selector "ul")* verranno rappresentati con un punto elenco di forma circolare e il testo in rosso, i secondi *(selector "ol")* con il testo in blu (Fig. 8). Un'ultima possibilità è quella di specificare lo stile di un elemento usando l'attributo *style* (CSS in linea):

```
<p style="color:red">red</p>
```

Ci sentiamo di sconsigliare vivamente l'uso dei CSS in linea, perché oltre a contravvenire la filosofia XML, che vuole una netta separazione tra stile e contenuto, rende difficoltosa la manutenzione e l'eventuale aggiornamento di uno stile (le modifiche devono essere fatte elemento per elemento in tutti i punti in cui compare …).

Domenico Pascuzzi

Fig. 8: Uso dei fogli di stile.

Java da zero: array e cicli.

Impariamo a ripeterci

Questo mese imparerai che quasi sempre un programma fa le stesse cose più di una volta, e scoprirai che esistono istruzioni e tipi di Java che sono fatti apposta. Ma prima…

Benvenuto nel nostro ristorante, gentile cliente. Ecco il menu di oggi.

Obiettivi di questa lezione:

- Imparerai cos'è la *visibilità* di una variabile.
- Imparerai a costruire dei *cicli*.
- Incontrerai una strana razza di variabili: gli *array*.

Tutti e tre i concetti sono assolutamente fondamentali in quasi tutti i linguaggi di programmazione. Quindi non perdiamo tempo e affrontiamo subito il primo.

TI VEDO, NON TI VEDO PIÙ

Abbiamo già incontrato le parentesi graffe nelle scorse puntate del nostro corso Java. Finora le abbiamo usate solo per delimitare il metodo *main()* e la classe che lo contiene:

```
class NomeDellaClasse {
  public static void main(String[] args) {
       // (programma)
   }
}
```

In realtà possiamo usare una coppia di parentesi in qualsiasi punto all'interno del *main()*. Una coppia di parentesi graffe vuota non ha molto senso, quindi di solito all'interno delle parentesi si trovano sempre una o più istruzioni Java. Queste istruzioni tra parentesi graffe costituiscono un *blocco* di istruzioni. A cosa serve un blocco di istruzioni? A due cose, e in questo articolo parleremo di entrambe. Ma per ora ci interessa solo la prima: qualsiasi variabile definita all'interno di un blocco è viva solo fino alla fine del blocco, e poi scompare. Si dice che la variabile ha *visibilità* solo all'interno del blocco nel quale è stata dichiarata (molti programmatori usano il termine inglese "scope" per indicare la visibilità). Tanto per complicare le cose, capita spesso di avere blocchi *annidati*, cioè uno dentro l'altro. Ad esempio:

```
class Scope {
  public static void main(String[] args) {
    int x = 1; {
      int y = 2;
      // riga 6: qui sono visibili sia x che y {
        int z = x + y;
        // riga 9: qui sono visibili x, y e z }
      // riga 11: ora z non e' piu' visibile
    }  // riga 12: qui e' visibile solo x
    y = y + 1;   // Errore di compilazione! }
}
```

Ciascuna variabile è visibile solo all'interno del più piccolo blocco che la contiene. Questo blocco si chiama *campo di visibilità* della variabile. Fuori dal suo campo di visibilità una variabile smette semplicemente di esistere, come se non l'avessimo mai dichiarata. Per questo motivo, se cerchi di compilare il programma qui sopra il compilatore Java ti darà un errore alla riga 13 ("cannot resolve symbol: variable y"). Questo è quanto devi sapere per ora sulla visibilità… Cioè giusto quanto basta per capire cos'è un *loop*.

REPETITA IUVANT

In qualsiasi linguaggio di programmazione capita spesso di dover ripetere più volte la stessa operazione, magari con qualche piccola variante. Ad esempio, immagina di avere una variabile che vale inizialmente 1. Vuoi stampare il suo valore, e poi incrementarlo di uno. Poi vuoi fare la stessa cosa una seconda volta. E una terza, e una quarta… Fino a quando la variabile non arriva al valore 5. E' chiaro che non sarebbe bello ripetere cinque volte le stesse due istruzioni:

Belle parentesi

Naturalmente ogni volta che apri una parentesi graffa per definire un blocco di istruzioni devi poi ricordarti di chiuderla. Abituati anche ad indentare il tuo codice facendolo rientrare sulla destra ogni volta che apri una parentesi graffa (prova a immaginare quanto sarebbe difficile leggere il codice del nostro primo esempio di questo mese se le istruzioni non fossero ben indentate).

Java

Se è così, allora...

Questo articolo ti spiega come usare i blocchi di istruzioni in un ciclo *while*. Naturalmente puoi usare lo stesso principio in qualsiasi altra istruzione di controllo di flusso, come ad esempio l'istruzione *if* (della quale abbiamo parlato il mese scorso). Ecco un pezzo di codice che "limita" il valore di una variabile in modo che non possa valere più di 10, e stampa sullo schermo una spiegazione di quel che sta facendo:

```
if(x > 10) {
  x = 10;
  System.out.println("Ho
      tagliato il valore
            a 10");
}
else {
  System.out.println("Il
     valore e' gia' OK");
}
```

Se la condizione dell'*if* è vera, allora vengono eseguite sia la seconda che la terza riga. Nota che le graffe dopo l'istruzione else sono facoltative, perché il blocco contiene una sola istruzione.

Esercizio 1

Riusciresti a scrivere un programma di una sola riga che si blocca in un ciclo infinito?
Pensaci un po'. Ti darò la soluzione alla fine dell'articolo. Intanto, ora che conosci i cicli possiamo finalmente parlare di array.

```
x = x + 1;
System.out.println("x vale " + x);
```

Per fortuna esistono alcuni modi per dire al sistema: "ripeti questa istruzione". Il più semplice è usare la parola chiave *while*.

while*(condizione)*
istruzione;

La condizione tra parentesi è una qualsiasi espressione booleana. La parola chiave *while* (che significa *"fintanto che"*) dice al sistema: *fintanto che* la condizione è *true*, esegui l'*istruzione*. Questo è quello che si chiama un *ciclo*, oppure (in inglese) un *loop*. Ad esempio, queste tre righe definiscono una variabile *x* con valore 1, e poi la incrementano ciclicamente *fintanto che* il suo valore resta inferiore o uguale 5:

```
int x = 1;
while(x <= 5)
  x = x + 1;
```

Nota che, all'uscita dal ciclo, *x* vale 6. Infatti, quando *x* vale 5 il programma entra nel ciclo per l'ultima volta ed esegue l'istruzione che incrementa *x*. A questo punto *x* vale 6, quindi la condizione diventa falsa e il programma esce dal ciclo. Per fortuna quello che vogliamo fare noi è un po' più interessante di questo codice: ogni volta che incrementiamo la variabile vogliamo stamparla.
Ora però abbiamo un problema. L'unica istruzione che viene ripetuta dal *while* è quella che segue immediatamente la condizione. Quindi se scrivessimo:

```
int x = 1;
while(x <= 5)
  x = x + 1;
  System.out.println(x);
```

otterremo di incrementare la variabile cinque volte, e solo alla fine stamparne il valore. Come facciamo a dire al *while* che vogliamo eseguire due istruzioni al posto di una? È a questo punto che ci vengono in aiuto i blocchi che hai imparato ad usare un paragrafo fa. Abbiamo detto che i blocchi hanno due caratteristiche. La prima è, come abbiamo visto, quella di definire i campi di visibilità delle variabili. La seconda è quella che un qualsiasi blocco di istruzioni si comporta come un'istruzione singola. Quindi possiamo risolvere il nostro problema con una semplice coppia di parentesi graffe:

```
int x = 1;
while(x <= 5) {
   x = x + 1;
   System.out.println(x);
   // entrambe le istruzioni precedenti
   // vengono eseguite 5 volte
}
```

Le due istruzioni tra parentesi graffe costituiscono il cosiddetto *corpo* del ciclo, cioè la sequenza di istruzioni che viene eseguita ciclicamente. Se vuoi che il corpo del ciclo sia composto da una sola istruzione, allora puoi fare a meno delle parentesi graffe (ma le puoi anche lasciare, se ti piacciono). Ricorda che non devi mettere nessun punto e virgola dopo la parentesi graffa chiusa. E' possibile che un ciclo non venga eseguito nemmeno una volta. Questo succede se la condizione risulta falsa quando il sistema incontra il ciclo:

```
int x = 0;
while(x > 5) {
  System.out.println("Questa roba non verra' mai
                                      stampata");
}
```

TANTI, E TUTTI FRATELLI

Ben presto scriverai dei veri programmi, e ti capiterà di avere moltissime variabili per le mani. Poniamo ad esempio che tu debba fare una lista dei primi cinque numeri pari. Ad esempio potresti aver voglia di stampare questa lista, o magari di sommare tutti e cinque gli elementi.
A questo punto siamo davanti ad un problema: che nome diamo alle variabili che conterranno questi numeri? E' chiaro che non possiamo inventare un nome per ciascuna variabile:

```
int primoNumeroPari = 2;
int secondoNumeroPari = 4;
int terzoNumeroPari = 6;
// ...e cosi' via
```

Se procedessimo in questo modo, prova ad immaginare quanto sarebbe noioso scrivere il codice che stampa e somma queste variabili. E se i numeri che vogliamo considerare fossero mille, cosa succederebbe? Sarebbe pressoché impossibile scrivere il codice, e se qualcuno ci provasse avrebbe praticamente la garanzia di sbagliare. Per fortuna Java, come quasi tutti i linguaggi, permette di creare delle *variabili indicizzate*, anche dette *array*. Un array è una variabile multipla. Puoi pensarlo come un elenco di variabili che hanno tutte lo stesso nome. Per distinguere le variabili si usa un indice, cioè un numero che indica la posizione della singola variabile nell'array. Ecco un esempio:

```
class NumeriPari {
  public static void main(String[] args) {
    int[] numeriPari;
    numeriPari = new int[5];
    int i = 0;
    while(i <= 4) {
      numeriPari[i] = (i + 1) * 2;
      i = i + 1;
    }
    int j = 0;
```

```
   while(j <= 4) {
          System.out.println(numeriPari[j]);
          j = j + 1;
      }
   }
}
```

Vediamo di studiare questo esempio riga per riga.

int[] numeriPari;
Questa riga è una dichiarazione (è come se scrivessimo *"int x"*). Dice che vogliamo un array di interi di nome *numeriPari*. Le due parentesi quadre subito dopo il tipo *int* specificano che vogliamo un array, non un singolo intero. Per ciascun tipo esiste l'array corrispondente, che è a sua volta un tipo. Quindi possiamo avere degli array di *double (double[])*, degli array di *boolean (boolean[])*, eccetera. Sì, è possibile avere anche array di array – ma per ora non ne parleremo. Andiamo avanti…

numeriPari = new int[5];
La riga precedente era la dichiarazione dell'array di interi *numeriPari*. Questa invece è la sua *inizializzazione*. Per inizializzare l'array dobbiamo dire esattamente quanto è lungo. La parola chiave *new* dice al sistema di creare un nuovo array, e il numero tra parentesi quadre ne decide la lunghezza. Dopo queste due istruzioni abbiamo un array di cinque elementi interi di nome *numeriPari*. Per fare riferimento a ciascun elemento ci basta scrivere il nome dell'array seguito dal suo indice tra parentesi quadre. Ad esempio:

```
System.out.println(numeriPari[1]);
```

questa istruzione stampa l'elemento di indice 1 dell'array. Ma attenzione: per ragioni storiche, la numerazione dell'indice di un array inizia sempre da 0, non da 1. Questo significa che l'elemento di indice 1 dell'array è il secondo, non il primo! Le righe successive dimostrano la potenza degli array. Naturalmente potremmo inizializzare ciascun elemento dell'array a mano, con del codice di questo tipo:

```
NumeriPari[0] = 2;
NumeriPari[1] = 4;
// ...eccetera
```

Ma se facessimo così non approfitteremmo del fatto di avere una variabile indicizzata. La natura degli array ("la morte loro", come direbbe un appassionato di cucina) è quella di essere usati nei cicli. Ecco come possiamo assegnare a ciascun elemento il suo valore iniziale:

```
int i = 0;
while(i <= 4) {
  numeriPari[i] = (i + 1) * 2;
  i = i + 1;
}
```

La variabile *i* viene usata sia come indice dell'array che come variabile di controllo per la condizione del ciclo. Questa variabile "conta" il numero di volte che entriamo nel ciclo, quindi la possiamo anche definire un *contatore*. La prima volta che il codice entra nel ciclo, *i* vale 0 (quindi è "minore o uguale a 4"). Nel ciclo viene assegnato il valore "zero più uno per due", cioè due, all'elemento di indice 0 dell'array. Poi l'indice/contatore viene incrementato e diventa uno. Quindi il programma entra nuovamente nel ciclo e assegna al secondo elemento dell'array *(numeriPari[1])* il valore "uno più uno per due", cioè quattro. E così via, per altre due volte. A un certo punto la variabile *i* raggiunge il valore 5, la condizione del *while* diventa falsa, e il programma esce dal ciclo. All'uscita dal ciclo, i quattro elementi del vettore sono stati inizializzati. Per stamparli possiamo usare un secondo ciclo, con un altro indice:

```
int j = 0;
while(j <= 4) {
   System.out.println(numeriPari[j]);
    j = j + 1;
}
```

Avrei potuto riutilizzare la variabile i assegnandole nuovamente il valore zero, ma non mi piace usare la stessa variabile più di una volta. Questo codice stampa tutti gli elementi del vettore, da *numeriPari[0]* a *numeriPari[4]*. Se fai girare il programma dovresti vedere questo output:

2
4
6
8
10

E ora facciamo il gioco del "cosa succederebbe se…".

FUORI DAI RANGHI

Cosa succederebbe se cercassimo di indicizzare l'elemento di indice 5 del nostro array? Dato che l'indice parte dal 0, gli elementi di *numeriPari* vanno da *numeriPari[0]* a *numeriPari[4]*. Quindi l'elemento *numeriPari[5]* non esiste. Se proviamo a referenziare un elemento che non esiste (usando un indice negativo, oppure un indice maggiore della lunghezza dell'array meno 1), il nostro programma può ancora essere compilato. In generale, il compilatore non è in grado di accorgersi che abbiamo sbagliato. Dopo tutto l'indice potrebbe non essere una semplice variabile o costante, ma il risultato di un calcolo molto complicato che il compilatore non può fare. L'errore arriva solo quando cerchiamo di far girare il programma.
Java è più "gentile" di altri linguaggi, che ci restituirebbero un valore imprevedibile. Inizializzare gli array elemento per elemento è un'operazione noiosa, quindi Java fa il lavoro per noi e inizializza ciascun elemento

Fuori i vettori

Qualcuno usa il nome italiano *vettori* per indicare gli array. In Java un "vettore" è un'altra cosa, quindi io preferisco usare sempre il nome inglese.

Esercizio 2

Prova a scrivere, compilare e far girare questo programma:

```
class IndiceTroppo
              Grande {
 public static void main(
         String[] args) {
   int[] unArray;
   unArray = new int[2];
   System.out.println(
         unArray[2]); } }
```

Quale errore ti dà la Java Virtual Machine? Un'altra domanda: cosa succederebbe se stampassimo un elemento di un array appena creato, prima di avergli assegnato un valore? Quale valore troveremmo?

Java

Java, linguaggio prudente

Il fatto che Java non permetta di accedere ad elementi di un vettore che non esistono può sembrare un cosa ovvia, ma è una caratteristica importante del linguaggio. In molti linguaggi (come in C) leggere e scrivere fuori dai limiti di un array è consentito, anche se si tratta di un errore. Il risultato sono errori di programmazione difficilissimi da individuare, con valori che apparentemente "cambiano da soli" - o peggio. Per fortuna in Java la Virtual Machine ci avvisa sempre appena cerchiamo di indicizzare un array fuori dai limiti consentiti.

Esercizio 3

Prova a scrivere, compilare e far girare questo programma:

```
class ElementoNon
        Inizializzato {
  public static void
    main(String[] args) {
    int[] unArray;
    unArray = new int[2];
    System.out.println(
            unArray[1]);
  }
}
```

Cosa viene stampato sullo schermo?

Esercizio 4

Prova a cambiare la dimensione dell'array da 5 a 100. Fai girare il programma.

ad un valore di default, che nel caso dei tipi numerici è zero (nel caso degli array di *boolean* è invece *false*). Ricorda che questo vale per gli array, non per le normali variabili: se dichiari una variabile *int* e cerchi di stamparla senza prima assegnarle un valore, otterrai un errore in fase di compilazione. Solo gli elementi degli array vengono inizializzati automaticamente.

UN PO' DI PULIZIA

Torniamo al programma *NumeriPari*, per vedere di ripulirlo un po'. Cominciamo dalle operazioni di incremento degli indici:

```
i = i + 1;
```

L'incremento è un'operazione talmente comune che Java ha un operatore apposta per dire la stessa cosa in modo più sintetico ed esplicito: l'*operatore di incremento*, che si scrive con un doppio +. Quindi l'istruzione precedente diventa semplicemente:

```
i++;
```

Un'altra cosa che possiamo fare è combinare la dichiarazione e l'inizializzazione dell'array in una sola istruzione, come facciamo di solito per le variabili "normali". Dopo tutto le due istruzioni devono sempre essere accoppiate, se non vogliamo che il compilatore ci dia un errore. Quindi possiamo scrivere semplicemente:

```
int[] numeriPari = new int[5];
```

Un'altra cosa che si può migliorare sono le condizioni "stupide" dei cicli. Cosa accadrebbe se volessimo modificare le dimensioni dell'array? Dovremmo andare a cambiare il codice in tre punti: nell'inizializzazione dell'array (e questo è il minimo che ci aspettiamo) e nelle condizioni dei due *while*. Sarebbe meglio se potessimo confrontare l'indice direttamente con la lunghezza dell'array, piuttosto che con una costante predeterminata. Per fortuna in Java è sempre possibile conoscere la lunghezza di un array scrivendone il nome seguito da "*.length*".
Ad esempio, questa istruzione stampa la lunghezza di un ipotetico array *v*:

```
System.out.println(v.length);
```

Quindi possiamo riscrivere i cicli *while* in modo tale che *i* vada da *0* al massimo indice dell'array, semplicemente chiedendo all'array qual è il suo massimo indice:

```
int i = 0;
while(i < numeriPari.length)
{
  // ...
}
```

Occhio a non sbagliare: l'operatore "minore o uguale" è diventato un semplice "minore". Se la lunghezza dell'array è 5, allora l'indice deve assumere al massimo il valore 4 (ricorda che gli indici partono da zero). E' facilissimo sbagliare le *condizioni limite* di un ciclo, ma ti abituerai a ragionare sempre sui limiti quando scrivi la condizione. Ecco la versione più elegante del nostro programmino:

```
class NumeriPari {
  public static void main(String[] args) {
    int[] numeriPari = new int[5];
    int i = 0;
    while(i < numeriPari.length) {
      numeriPari[i] = (i + 1) * 2;
      i++;
    }
    int j = 0;
    while(j < numeriPari.length) {
      System.out.println(numeriPari[j]);
      j++;
    }
  }
}
```

RACCOGLIENDO LE NOSTRE COSE

Come al solito, prima di concludere devo ancora darti la soluzione di un esercizio. Ti avevo chiesto di scrivere un programma di una sola istruzione che si blocca in un ciclo infinito. Come avviene per molti altri problemi, la soluzione è semplicissima una volta che la si conosce:

```
class CicloInfinito {
public static void main(String[] args) {
       while(true) {}
  }
}
```

La condizione del *while* è sempre vera, e nel "corpo" del ciclo non succede niente che la renda falsa (in realtà non succede niente *punto e basta*). Quindi il ciclo va avanti per sempre, o meglio fino a quando non premi la combinazione di tasti *Ctrl+C* per terminare il programma. Puoi anche rimuovere le parentesi graffe del *while* (che definiscono un blocco di istruzioni vuoto) e sostituirle con una singola istruzione vuota, cioè un punto e virgola:

```
while(true);
```

A questo punto devo darti appuntamento al mese venturo. E' un peccato, perché abbiamo ancora tanto da dire sui cicli e sugli array. Consolati pensando che nel prossimo articolo ti aspetta qualche esercizio difficile e interessante. A presto!

Paolo Perrotta

Inserimento dei dati nelle form.

Caselle di riepilogo e caselle combinate

I controlli ListBox, CheckedListBox e ComboBox forniscono all'utente maggiori possibilità di scelta occupando meno spazio rispetto ai controlli descritti nel numero precedente.

La volta scorsa abbiamo visto come impedire all'utente di introdurre informazioni errate, attraverso una serie di controlli da utilizzare laddove i possibili dati da inserire siano in numero limitato. In questo articolo descriveremo un'altra serie di controlli utilizzabili per proporre all'utente una serie di opzioni diverse: i controlli *ListBox* (caselle di riepilogo), *CheckedListBox* e *ComboBox* (caselle combinate). Porteremo a termine, inoltre, la descrizione dei comportamenti comuni a tutti i controlli, iniziata negli articoli precedenti, descrivendo gli eventi comuni.

EVENTI COMUNI

Prima di passare alla descrizione dei controlli in esame analizziamo alcuni degli eventi comuni a tutti i controlli, divisi per categoria.

Focus - Gli eventi di attivazione si verificano nel seguente ordine:

- *Enter.* Generato quando viene immesso il controllo.
- *GotFocus.* Generato quando il controllo riceve lo stato attivo.
- *Leave.* Generato quando lo stato attivo esce dall'area del controllo.
- *Validating.* Generato quando il controllo viene convalidato.
- *Validated.* Generato al termine della convalida del controllo.
- *LostFocus.* Il controllo perde lo stato attivo.

Tastiera e Mouse

- *Click*: L'utente fa click con il mouse sul controllo.
- *DoubleClick*: Generato quando l'utente fa doppio click sul controllo.
- *MouseDown* Generato quando l'utente preme un pulsante del mouse. Riceve i valori *Button*, *Clicks*, *Delta*, *X*, *Y*.
- *MouseUp* L'utente rilascia un pulsante del mouse, riceve gli stessi valori di *MouseDown*.
- *MouseMove* Quando il mouse è stato posizionato sul controllo, viene generato ad ogni movimento del mouse.
- *KeyDown* Generato quando è stato premuto un tasto mentre il controllo aveva il focus.
- *KeyUp* Generato quando è stato rilasciato un tasto mentre il controllo aveva il focus.
- *KeyPress* Viene generato quando è stato premuto un tasto stampabile mentre il controllo aveva il focus.

Aspetto

- *Paint* Generato quando il controllo viene ridisegnato.
- *Move* Generato quando il controllo viene spostato.
- *Resize* Il controllo viene ridimensionato.

GESTIONE DEL FOCUS

VB .NET genera sei eventi diversi quando un controllo riceve o perde lo stato attivo (Focus), per stato attivo s'intende la capacità di un controllo di ricevere l'input dell'utente tramite il mouse o la tastiera, in ordine sono: *Enter, GotFocus, Leave, Validating, Validated, LostFocus.* Gli eventi *Enter* e *GotFocus* vengono generati quando un controllo riceve lo stato attivo, mentre gli eventi *Leave, Validating, Validated, LostFocus* si verificano nell'ordine quando il controllo perde lo stato attivo e passa ad un controllo diverso. Gli eventi *Validating* e *Validated* sono scatenati soltanto se la proprietà *CausesValidation* è impostata a *True*, sia nel controllo che possiede il focus sia in quello che cerca di ottenerlo. Generalmente l'evento *Validating* è usato per intercettare i valori non validi di un campo. Se il valore di un controllo non è del tipo previsto dal codice, si deve semplicemente impostare la proprietà *Cancel* dell'argomento al valore *True*. Per la gestione del focus, si possono utilizzare (oltre alle proprietà *Enabled* , *TabIndex* e *TabStop* descrit-

Ricerche

Il metodo *FindString* consente di trovare il primo elemento, nei controlli *ComboBox* e *ListBox*, che inizia con la stringa specificata.

Stato attivo

Per stato attivo s'intende la capacità di un controllo di ricevere l'input dell'utente tramite il mouse o la tastiera.

Visual Basic .NET

te nei precedenti articoli) due nuove proprietà:

- La proprietà **CanFocus** consente di determinare se un dato controllo è in grado di ottenere il focus di input (cosa che accade quando entrambe le proprietà *Visible* ed *Enabled* risultino impostate a *True*).
- **Focused** indica se il controllo possiede il focus.

KeyPress, KeyDown, KeyUp

Passiamo ora agli eventi che ci permettono di gestire la tastiera e che vengono generati quando l'utente preme e rilascia un tasto sul controllo attivo. Quando l'utente preme un tasto viene generato l'evento *KeyDown*, VB traduce il tasto nel corrispondente carattere ASCII nell'evento *KeyPress*, ed infine quando il tasto viene rilasciato viene generato l'evento *KeyUp*. L'evento *KeyPress* riceve due argomenti: l'argomento sender che rappresenta l'oggetto che ha scatenato l'evento e l'argomento e che rappresenta un oggetto *KeyPressEventArgs*. L'oggetto *KeyPressEventArgs* espone solo due proprietà:

- **KeyChar** corrisponde al carattere equivalente al tasto premuto.
- **Handled** rappresenta un valore booleano che viene impostato a *True* per notificare al motore della form l'avvenuta elaborazione dell'evento e che non dovrebbe accettare ulteriori azioni per quanto riguarda il tasto premuto.

la proprietà *Handled* è un valore booleano al quale bisognerebbe assegnare *True* nel caso lo sviluppatore voglia gestire in proprio l'evento. Per definire, ad esempio, un campo che ammette solo testo, eliminando la possibilità di inserire valori numerici, si può sfruttare la proprietà *Handled* scrivendo semplicemente:

```
Private Sub TextBox1_KeyPress(ByVal sender As Object,
   ByVal e As System.Windows.Forms.KeyPressEventArgs)
                                   Handles TextBox1.KeyPress
   'Se il carattere è una cifra notifica al motore della form
   'di Ignorarlo.
      If IsNumeric(e.KeyChar) Then
          e.Handled = True
      End If
End Sub
```

Gli eventi *KeyDown* e *KeyUp* ricevono gli stessi argomenti: l'argomento sender che rappresenta l'oggetto che ha scatenato l'evento e l'argomento e che rappresenta un oggetto *KeyEventArgs*. L'oggetto *KeyEventArgs* espone le seguenti proprietà:

- **Alt, Shift, Control** restituiscono un valore pari a *True* se sono stati premuti rispettivamente i tasti *ALT*, *MAIUSC* o *CTRL*.
- **Modifiers** restituisce il valore, codificato in bit, della combinazione di tasti di modifica (*ALT*, *MAIUSC* o *CTRL*) premuta dall'utente.
- **KeyCode** restituisce il codice (indicato sotto forma di un valore enumerato *Keys*) del tasto che viene premuto o rilasciato.
- **KeyValue** restituisce il codice numerico del tasto che viene premuto o rilasciato.
- **KeyData** restituisce il codice del tasto premuto, comprensivo degli eventuali flag di modifica, che indicano la pressione contemporanea (combinata o singola) dei tasti *CTRL*, *MAIUSC* e *ALT*.
- **Handled** ottiene o imposta un valore booleano che indica se l'evento è già stato gestito.

IL CONTROLLO LISTBOX

Così come il controllo *ComboBox*, che analizzeremo in un secondo momento, il controllo *ListBox* visualizza una lista di opzioni (o valori) selezionabili dall'utente. Le proprietà più interessanti sono:

- La proprietà **MultiColumn**, utilizzata per indicare se il *ListBox* deve essere formato da più colonne. Per default *MultiColumn* è pari a *False* e visualizza un elenco a colonna unica con la barra di scorrimento verticale, ponendo *MultiColumn* a *True* si avrà un elenco a due (o più) colonne con barra di scorrimento orizzontale della dimensione specificata in *ColumnWidth*
- La proprietà **Sorted** ha gli stessi effetti per il *ComboBox*, ponendola pari a *True* le voci contenute nel controllo saranno automaticamente ordinate in ordine alfabetico discendente, (dalla A alla Z) senza scrivere una riga di codice.
- La proprietà **SelectionMode** permette di impostare le modalità di selezione degli elementi di un controllo *ListBox*. Si possono definire il numero massimo di elementi nel controllo *ListBox* per ciascuna selezione e le modalità per eseguire selezioni multiple. Impostando la proprietà a *MultiExtended* si può eseguire la multiselezione come normalmente si fa in ambiente Windows (utilizzando il tasto *Shift* e le frecce direzionali, oppure il tasto *CTRL*). Impostando la proprietà a *MultiSimple*, per selezionare o deselezionare un elemento dell'elenco, si deve cliccare con il mouse oppure premere la barra spaziatrice. Con il valore predefinito *One*, è possibile selezionare un solo elemento per volta.

AGGIUNGERE VALORI ALLA LISTA

In fase di progettazione, è possibile aggiungere valori in un *ListBox* ed in un *ComboBox*, utilizzando la proprietà (collezione) *Items* dalla finestra delle proprietà, per questo si deve:

- Selezionare il controllo e visualizzare la finestra delle proprietà.
- Cliccare sul pulsante con i puntini di sospensione posto accanto alla proprietà *Items*, apparirà la fine-

Aggiornare

I metodi *BeginUpdate* e *EndUpdate* possono essere utilizzati nel caso in cui si devono aggiungere un numero elevato di elementi ai controlli *ComboBox* e *ListBox* senza che questi vengano ridisegnati ogni volta che viene aggiunto un elemento all'elenco. Questa modalità di aggiunta di elementi evita che si verifichi uno sfarfallio durante il disegno dei controlli.

BeginUpdate impedisce il ridisegno del controllo quando vengono aggiunti elementi uno per volta con il metodo *Add*

EndUpdate, una volta completata l'aggiunta di elementi all'elenco, permette di riprendere il disegno dei controlli dopo la sospensione eseguita dal metodo *BeginUpdate*.

stra *Editor* dell'insieme *String*.

- Immettere l'elenco di stringhe che dovranno apparire nel controllo una per riga ed infine premere sul pulsante *OK*.

Fig. 1: E' possibile popolare un listbox durante la progettazione.

In fase di esecuzione si può utilizzare la collezione *Items*, che raccoglie in sé tutti gli elementi del controllo. *Items* è un oggetto *Icollection*, pertanto è possibile gestire tutti i suoi elementi utilizzando gli usuali metodi *Add*, *Remove* e *Clear*. Generalmente il controllo viene popolato nella procedura evento *Load* della form, ad esempio:

```
Private Sub Form1_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
     ListBox1.Items.Add("Lunedi")
     ListBox1.Items.Add("Martedi")
     ListBox1.Items.Add("Mercoledi")
 `.................
End Sub
```

Se la proprietà *Sorted* del controllo *ListBox* è impostata su *true*, l'elemento viene inserito nell'elenco in ordine alfabetico. In caso contrario, viene inserito alla fine dell'elenco. A questa collezione è possibile aggiungere qualsiasi tipo di oggetto, e non semplicemente delle stringhe. La proprietà *ItemData* non è più gestita, ma la possibilità di memorizzare oggetti la rende effettivamente superflua. Per eliminare voci dall'elenco è possibile usare i seguenti metodi:

- **Remove** elimina le voci una per volta. Consente di eliminare l'oggetto specificato in qualunque posizione si trovi. *ListBox1.Items.Remove("Lunedi")*. Se l'oggetto (dal nome *"Lunedi"* nell'esempio) non esiste, non viene generata alcuna eccezione.
- **RemoveAt** elimina le voci una per volta. Consente di eliminare l'elemento nella posizione specificata (il primo elemento ha indice zero). *ListBox1.Items.RemoveAt(3)*. Se la posizione specificata non esiste, viene generata un'eccezione.
- **Clear** elimina tutte le voci dell'elenco. Per eliminare l'intero elenco è sufficiente scrivere: `ListBox1.Items.Clear()`.
  Per recuperare l'elemento selezionato nella *ListBox* si possono utilizzare le proprietà:
  - **SelectedItem** per ottenere l'elemento selezionato.
  - **SelectedIndex** per ottenere l'indice dell'elemento selezionato.

CHECKEDLISTBOX

Il controllo *CheckedListBox* è in pratica un controllo *ListBox* che riporta una casella di controllo alla sinistra di ciascun elemento. In VB6 corrisponde al controllo *ListBox* con la proprietà *Style* impostata a *1-CheckBoxes*. Questo controllo consente all'utente di contrassegnare uno o più elementi semplicemente cliccando con il mouse sulla casella. Tuttavia, la *CheckedListBox* non è in grado di gestire la selezione multipla, pertanto la proprietà *SelectionMode* serve solo a impostare o ad impedire la selezione di un singolo elemento. Per lo stesso motivo, le collezioni *SelectedItems* e *SelectedIndices* sono state sostituite rispettivamente da *CheckedItems* e *CheckedIndices*. Il controllo *CheckedListBox* espone due nuove proprietà:

- La proprietà **ThreeDCheckBoxes** può essere impostata a *True* per mostrare caselle di controllo tridimensionali. Il valore di default è *False*.
- La proprietà **CheckOnClick**, può essere impostata a *True*, per consentire all'utente di contrassegnare o meno gli elementi con un singolo clic. Il valore di default è *False*. Il comportamento predefinito consiste nel modificare la selezione in occasione del primo clic e nell'applicare il segno di spunta solo quando l'utente fa di nuovo clic.

COMBOBOX

Nel *ComboBox*, a differenza del *ListBox*, la lista di voci è una lista a scomparsa ed è visibile soltanto se si clicca sulla freccia rivolta verso il basso a destra del controllo. Nel *ComboBox* è, inoltre, possibile digitare un valore che non compare nella lista di quelli disponibili. Il controllo *ComboBox* si comporta in maniera simile al *ListBox*, restano per questo validi i concetti esposti in precedenza. Si può quindi popolare in fase di progettazione utilizzando la proprietà (collezione) *Items* dalla finestra delle proprietà, oppure in fase di esecuzione da codice utilizzando la collezione *Items*.

```
Private Sub Form2_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
     ComboBox1.Items.Add("Lunedi")
     ComboBox1.Items.Add("Martedi")
     ComboBox1.Items.Add("Mercoledi")
 `.................
End Sub
```

La proprietà tipica del *ComboBox* è la proprietà *DropDownStyle* che può assumere i seguenti valori:

- **DropDown** è il valore di default che permette di visualizzare la classica casella di riepilogo a discesa ed una casella di testo. Sarà possibile selezionare

Visual Basic .NET

Possibili valori di SelectionMode

MultiExtended - **È possibile selezionare più elementi utilizzando i tasti *MAIUSC, CTRL* ed i tasti di direzione per effettuare selezioni.**

MultiSimple - **È possibile selezionare più elementi.**

One - **È possibile selezionare solo un elemento.**

None - **Non è possibile selezionare nessun elemento.**

CheckedIndices

La collezione permette di recuperare gli indici degli elementi selezionati in un controllo *CheckedListBox.*

Selezione

L'evento *SelectedIndexChanged,* comune a tutti e tre i controlli descritti nell'articolo, viene generato quando si seleziona un elemento nella lista, in altre parole quando si modifica la proprietà *SelectedIndex* del controllo.

una voce della casella di riepilogo oppure digitare un valore non compreso nell'elenco.

- **Simple** permette di visualizzare un *ComboBox* in cui la lista delle voci è sempre visibile, in sostanza una casella di testo più una *ListBox*. Per default la casella è dimensionata in modo da visualizzare solo la casella in cui digitare il valore, senza che nessun elemento dell'elenco sia visualizzato.
- **DropDownList** visualizza graficamente il *ComboBox* come lo stile *DropDown*, ma consente soltanto la selezione dall'elenco a discesa e non permette l'immissione di testo libero.

APPLICAZIONE DI ESEMPIO

Per meglio fissare i concetti espressi finora modifichiamo l'applicazione del precedente articolo utilizzando i controlli appena descritti. Ricordiamo che l'applicazione deve mostrare un prospetto riassuntivo delle condizioni meteorologiche verificatesi durante la settimana. Per il nostro scopo avremo bisogno di:

- Un *ComboBox* con i giorni della settimana (*ComboBoxGiorni*)
- Una *CheckedListBox* con le condizioni atmosferiche (*CheckedListBoxTempo*)
- Un *Button* per confermare la condizione atmosferica verificatasi, dal nome *ButtonConferma*
- Un *TextBox* multiriga che visualizzi il prospetto riassuntivo, dal nome *TextBoxRiepilogo*.

Nell'evento *Load* della form scriviamo il codice necessario a popolare i due controlli, in particolare chiamiamo le due procedure *InizializzaComboBox* e *InizializzaCListBox* che, utilizzando il metodo *Add* della collezione *Items*, si preoccuperanno di riempire effettivamente i controlli:

```
Private Sub Form1_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
    InizializzaComboBox()
    InizializzaCListBox()
End Sub
Private Sub InizializzaComboBox()
    ComboBoxGiorni.Items.Add("Lunedi")
    ComboBoxGiorni.Items.Add("Martedi")
    ComboBoxGiorni.Items.Add("Mercoledi")
    ComboBoxGiorni.Items.Add("Giovedi")
    ComboBoxGiorni.Items.Add("Venerdi")
    ComboBoxGiorni.Items.Add("Sabato")
    ComboBoxGiorni.Items.Add("Domenica")
    ComboBoxGiorni.SelectedIndex = 0
End Sub
Private Sub InizializzaCListBox()
    CheckedListBoxTempo.Items.Add("Sole")
    CheckedListBoxTempo.Items.Add("Pioggia")
    CheckedListBoxTempo.Items.Add("Neve")
    CheckedListBoxTempo.Items.Add("Vento")
End Sub
```

Il codice necessario a visualizzare il giorno e le condizioni atmosferiche selezionate, dovrà essere scritto nell'evento *Click* di *ButtonConferma* e rimane uguale a quello del precedente articolo

```
Private Sub ButtonConferma_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs)
                        Handles ButtonConferma.Click
    Dim Giorno As String
    Dim Tempo As String
    Giorno = DeterminaGiorno()
    Tempo = DeterminaTempo()
    TextBoxRiepilogo.Text = TextBoxRiepilogo.Text
                        + vbCrLf + Giorno + Tempo
End Sub
```

Sono state dichiarate due variabili, *Giorno* e *Tempo*, di tipo stringa. Alle due variabili si assegna il valore di ritorno delle due funzioni *DeterminaGiorno* e *DeterminaTempo*, che si faranno carico di comporre la stringa risultato in base alle scelte dell'utente.
Infine nel *TextBox* di riepilogo si mostrano le informazioni su ogni singola riga, separandole dalla combinazione dei caratteri di ritorno a capo e avanzamento riga *(Chr(13) + Chr(10))* rappresentati dalla costante *vbCrLf*. La funzione *DeterminaTempo* dovrà scorrere gli elementi selezionati nel controllo *CheckedListBox* utilizzando la collezione di elementi selezionati, *CheckedItems*, e comporre la stringa informativa aggiungendo la condizione atmosferica corrispondente al valore selezionato.

```
Private Function DeterminaTempo() As String
  Dim TmpStringa, Tempo As String
  TmpStringa = ""
  For Each Tempo In CheckedListBoxTempo.CheckedItems
    TmpStringa = TmpStringa + " " + Tempo
  Next
  DeterminaTempo = TmpStringa
End Function
```

La funzione *DeterminaGiorno* dovrà recuperare l'elemento selezionato nel *ComboBox* utilizzando la proprietà *SelectedItem*

```
  Private Function DeterminaGiorno() As String
    DeterminaGiorno = ComboBoxGiorni.SelectedItem + ": "
  End Function
```

CONCLUSIONI

Negli ultimi due articoli, abbiamo descritto una serie di semplici controlli che permettono di presentare all'utente una serie di opzioni di scelta, facilitando l'immissione di informazioni in un sistema informatico. Nei prossimi articoli analizzeremo ulteriori controlli che facilitano la vita all'utente e rendono più attraente l'interfaccia.

Ing. Luigi Buono

C#

Le classi astratte e la superclasse object.

Ereditarietà nelle classi astratte

Siamo giunti all'ultima delle tre lezioni che, in questo corso, sono state dedicate all'ereditarietà. Dopo aver esaminato le caratteristiche salienti dell'ereditarietà con C#, questo mese chiudiamo passando in rassegna le classi astratte e la superclasse object. Le nozioni qui presentate torneranno presto utili: saranno un trampolino di lancio verso lo studio di altre caratteristiche del linguaggio e della piattaforma .NET.

Nel corso della lezione precedente sono state viste alcune tra le più importanti caratteristiche dell'ereditarietà con C# e .NET. In particolare, si è visto come definire i costruttori nelle classi derivate, come attuare il nascondimento dei nomi e come ridefinire i metodi ereditati. Inoltre, abbiamo appreso come far uso delle classi derivate attraverso dei riferimenti alle loro superclassi, nozione importantissima per lo sviluppo di moltissime applicazioni. Questo mese proseguiamo e portiamo a termine il nostro studio concernente l'ereditarietà, esplorando le classi virtuali ed esaminando la classe *object*, con le norme che ne regolano l'utilizzo.

CLASSI ASTRATTE E METODI ASTRATTI

Le classi astratte sono classi create appositamente per essere derivate. Una classe astratta può essere derivata, ma non è possibile avere delle sue istanze da impiegare direttamente in un software. Alla base dell'astrazione di una classe è la parola chiave *abstract*. Una classe, pertanto, è astratta quando è dichiarata secondo il modello:

```
abstract class NomeClasse {
  // ...
}
```

Le classi astratte contengono sempre uno o più *metodi astratti*. Un metodo è astratto quando:

- La sua dichiarazione è preceduta dalla parola chiave *abstract*.
- Non ha corpo.

Le classi astratte sono impiegate per definire un modello di comportamento, che ogni sottoclasse dovrà rispettare, implementando tutti i metodi astratti in essa contenuti. Prendiamo in esame la classe astratta *Animale*:

```
abstract class Animale {
  public void mangia() {
    System.Console.WriteLine("Gnam!");
  }
  public abstract void faiVerso();
}
```

La classe contiene due metodi: *mangia()* e *faiVerso()*. L'implementazione del primo è molto semplice da realizzare. Ma cosa dobbiamo mettere dentro *faiVerso()? Animale* è una classe che esprime un concetto molto generico e assai poco concreto. Che verso fa un animale? La risposta dipende dal tipo di animale. Un gatto fa "miao", un cane "bau"... ma un animale in senso generico?
Non è possibile stabilirlo! Per questo motivo, la classe *Animale* è stata dichiarata astratta, ed il suo metodo *faiVerso()* non è stato definito. Di conseguenza, non è possibile istanziare oggetti della classe *Animale*.

```
Animale a = new Animale(); // Sbagliato!
```

C#

Un'istruzione come questa porta ad un errore di compilazione:

```
Impossibile creare un'istanza della classe o
                    dell'interfaccia abstract "Animale".
```

Animale esiste solo per essere derivata. Ogni suo sottoclasse dovrà, obbligatoriamente, dare definizione al metodo *faiVerso()*, tranne nel caso non sia essa stessa una classe astratta, pena un errore di compilazione. Ecco alcune simpatiche implementazioni:

```
class Gatto : Animale {

  public override void faiVerso() {
    System.Console.WriteLine("Miaaaaaaaaooo!");
  }

}

class Cane : Animale {

  public override void faiVerso() {
    System.Console.WriteLine("Bau!");
  }

}

class Topo : Animale {

  public override void faiVerso() {
    System.Console.WriteLine("Squit!");
  }

}
```

Verifichiamo la funzionalità delle tre classi appena realizzate:

```
class Test {

  public static void Main() {
    Animale[] zoo = new Animale[3];
    zoo[0] = new Gatto();
    zoo[1] = new Cane();
    zoo[2] = new Topo();
    for (int i = 0; i < zoo.Length; i++)
      zoo[i].faiVerso();
  }
}
```

L'output è il seguente:

Miaaaaaaaaooo!
Bau!
Squit!

Si osservi come un metodo astratto sia automaticamente un metodo virtuale. D'altra parte, non potrebbe essere diversamente.
Come vedremo prossimamente, le interfacce sono una completa estremizzazione del concetto di classe astratta.

LA CLASSE OBJECT

In C#, come si è detto due lezioni fa, ogni classe che non estende esplicitamente un'altra classe, eredita direttamente ed implicitamente dalla classe *object*, definita nella libreria base della piattaforma .NET. Il nome completo di questa classe, nel framework di .NET, è proprio *System.Object*. Questa regola vale sia per le normali classi che abbiamo studiato sinora, che vanno a creare dei tipi riferimento, sia per i tipi valore. Quindi, anche un valore *int* è effettivamente derivato da *object*. Tutto, ma proprio tutto, deriva da *object*.
Questa è una delle novità che maggiormente contraddistinguono C# da altri linguaggi come Java, in cui almeno i tipi valore non sono considerati degli oggetti. Ne conseguono due importanti caratteristiche della programmazione C#:

- Una variabile di tipo *object* può far riferimento ad un oggetto di qualsiasi tipo, sia esso un tipo riferimento o, indifferentemente, un tipo valore. Anche un array può essere memorizzato in un riferimento di tipo *object*.

- La classe *object* definisce un certo numero di metodi, che sono quindi automaticamente disponibili in ogni oggetto.

La prima caratteristica può essere facilmente osservata:

```
class Test {

  public static void Main() {
    string s = "Ciao";
    int i = 10;
    double d = 3.14;
    Test t = new Test();
    int[] array = {1, 2, 3};

    object[] o = new object[5];
    o[0] = s;
    o[1] = i;
    o[2] = d;
    o[3] = t;
    o[4] = array;
  }

}
```

Cos'è un codice hash?

Un codice hash è una chiave che identifica univocamente il contenuto di un oggetto. Il calcolo del codice hash associato ad un oggetto consiste nel desumere un intero da ogni proprietà dell'oggetto stesso, secondo regole che possono variare da un caso ad un altro. Due oggetti equivalenti, qualunque sia la nozione di equivalenza in uso, devono corrispondere allo stesso codice hash. I codici hash vengono usati soprattutto durante lo sviluppo delle collezioni che comprendono funzionalità per la ricerca rapida: identificare un oggetto mediante un intero è molto più efficiente che non attraverso delle stringhe o altri tipi di campi. I confronti, con i codici hash, sono molto meno dispendiosi di risorse di calcolo.

Sono stati creati cinque differenti oggetti, alcuni di tipo riferimento, altri di tipo valore. Quindi, è stato creato un vettore di *object*, e ad ognuno dei cinque elementi è stata associata una delle variabili precedentemente definite. L'operazione riesce alla perfezione, senza far segnalare alcun errore di compilazione. Una variabile *object* è così generica da poter memorizzare oggetti di ogni tipo.
I metodi definiti da *object* sono mostrati di seguito:

- **public virtual bool Equals(object oggetto)**
  Stabilisce se l'oggetto di invocazione è lo stesso al quale fa riferimento l'argomento.
- **public static bool Equals(object ogg1, object ogg2)**
  Stabilisce se i due argomenti forniti fanno riferimento allo stesso oggetto.
- **public virtual int GetHashCode()**
  Calcola e restituisce il codice hash associato all'oggetto di invocazione.
- **public Type GetType()**
  Fornisce informazioni sul tipo dell'oggetto di invocazione.
- **protected object MemberwiseClone()**
  Crea una "copia vuota" dell'oggetto di invocazione. Una copia vuota consiste in un oggetto identico al precedente, ottenuto copiando tutti i suoi membri, ma non gli oggetti ai quali tali membri fanno riferimento.
- **public static bool ReferenceEquals (object ogg1, object ogg2)**
  Stabilisce se i due argomenti forniti fanno riferimento allo stesso oggetto.
- **public virtual string ToString()**
  Fornisce una rappresentazione in stringa dell'oggetto di invocazione.

Come è possibile osservare, alcuni dei metodi sopra riportati sono virtuali. Quindi, possono essere ridefiniti per meglio adattarsi alle esigenze delle specifiche classi. Ad esempio, spesso si è soliti ridefinire il metodo *Equals()*. La versione suggerita da *object*, infatti, non fa altro che mettere a confronto i due riferimenti in gioco.
In situazioni più peculiari, però, l'equivalenza di due oggetti può avere significato più vasto. Anche *ToString()* è un metodo spesso ridefinito, così come lo è *GetHashCode()* nei casi in cui l'oggetto dovrà essere usato in situazioni in cui i codici hash giocano un ruolo ben preciso.

BOXING E UNBOXING

Abbiamo già detto che, in C#, anche i tipi valore sono degli oggetti, derivati dalla classe *object*. Nell'esempio di codice riportato nel paragrafo precedente, a dimostrazione di quanto è stato affermato, alcuni oggetti di tipo valore sono stati riassociati a variabili di tipo *object*:

```
// ...
int i = 10;
double d = 3.14;
// ...
object[] o = new object[5];
// ...
o[1] = i;
o[2] = d;
// ...
```

Quando si compie un'operazione di questo tipo, C# esegue automaticamente un'operazione che va sotto il nome di *boxing* (letteralmente, *mettere in scatola*). Il boxing consiste nel passaggio da una politica di gestione per valore ad una per riferimento.
L'operazione è automatica, ed il programmatore non deve curarsene. Il passaggio inverso, chiamato *unboxing*, si realizza attraverso un semplice casting:

```
int i1 = 5;
object o = i1;
int i2 = (int)o; // Unboxing
```

Il boxing permette delle operazioni che in altri linguaggi apparirebbero prive di senso:

```
System.Console.WriteLine(5.ToString());
                              // Perfettamente lecito!
```

CONCLUSIONI

Termina qui lo studio dell'ereditarietà in C#. Molte delle nozioni acquisite nel corso degli ultimi mesi, oltre ad essere effettivamente utili per lo sviluppo di un vasto numero di applicazioni, torneranno ancora nel nostro percorso di studio.
Già con le interfacce, il mese prossimo, rivisiteremo il concetto di classe astratta. Vi consiglio, inoltre, di consolidare quanto più possibile le nozioni recentemente esposte, giacché gran parte della libreria di .NET fa ampio uso dei concetti di ereditarietà.
Aver ben chiaro quanto illustrato significa, di fatto, muoversi con maggiore abilità tanto nello sviluppo di nuovi tipi di dati quanto nell'utilizzo di quelli offerti dall'ambiente di runtime di .NET.

Carlo Pelliccia

Bibliografia

- **GUIDA A C#**
  Herbert Schildt
  (McGraw-Hill)
  ISBN 88-386-4264-8
  2002
- **INTRODUZIONE A C#**
  Eric Gunnerson
  (Mondadori Informatica)
  ISBN 88-8331-185-X
  2001
- **C# GUIDA PER LO SVILUPPATORE**
  Simon Robinson e altri
  Hoepli, 2001
  ISBN 88-203-2962-X

Le librerie del linguaggio.

Librerie standard del C++: le Stringhe

Dopo l'introduzione della precedente puntata, approfondiamo alcuni aspetti utili delle stringhe nella libreria standard del C++.

Nel corso della puntata precedente abbiamo appreso la parola magica in grado di migliorare la qualità della vita di noi poveri programmatori C++:

using namespace std;

Mediante questa formula si ha accesso alle potenti funzionalità della libreria standard del C++, che ci mette a disposizione classi e funzioni dai molteplici utilizzi: una vera miniera di codice sicuro ed ottimizzato, quindi riutilizzabile a cuor leggero nei nostri programmi.
Abbiamo iniziato il nostro viaggio di esplorazione dall'argomento che più turba chi non conosce la parola magica detta poc'anzi: la manipolazione delle stringhe. Abbiamo visto come la libreria standard del C++ ci metta a disposizione un tipo apposito, il tipo *string*: in questo modo le stringhe non sono più scomodi array di caratteri chiusi dal terminatore di stringa (di cui non dovremo più curarci), convenzionalmente usati nei programmi C-like.
Il fatto che le stringhe siano trasposte in un vero e proprio tipo ha molteplici vantaggi, primo tra tutti il fatto che possiamo disinteressarci di come una stringa sia strutturata al suo interno. Non dobbiamo più pensare ad una stringa come ad un array di caratteri: banalmente, una stringa è una stringa. Ne abbiamo avuto una prova guardando come si comporta il compilatore quando usiamo i soliti operatori con variabili di questo tipo: la maggior parte di essi è opportunamente ridefinita, e questo non ci fa distinguere una stringa da uno dei tipi predefiniti.

LE DIMENSIONI CONTANO: SIZE() E CAPACITY()

Se immaginassimo una stringa come un array di caratteri, allora non potremmo che figurarci un'altra (poco allettante) realtà: dovremo gestire da soli i dimensionamenti delle stringhe, allocando nuova memoria ogni volta che ciò si rendesse necessario, ed effettuando ovunque i necessari controlli atti a mantenere la consistenza del codice; questo è esattamente quello che abbiamo fatto di solito negli esempi delle puntate passate, mediante l'uso di funzioni di libreria come la *strlen()* e la *strcpy()*.
Invece, una *string* è a tutti gli effetti un oggetto, con le sue proprietà e i suoi metodi, e la sua gestione interna è del tutto automatizzata: il programmatore non ha bisogno di gestirne le dimensioni (e questa cosa è particolarmente apprezzabile quando la dimensione varia spesso ed in maniera imprevedibile), così come non deve curarsi della maniera in cui è implementata, in piena filosofia Object-Oriented.
Una stringa contiene un buffer per i dati, che altro non è che l'insieme dei caratteri che la costituiscono (l'array di *char*, per intenderci). Contiene poi dei dati che possono tornare utili, come il numero dei caratteri che costituiscono la stringa (cioè, le posizioni occupate all'interno del buffer dei dati), oppure il numero di caratteri massimo che il buffer può contenere in un determinato momento (cioè, a tutti gli effetti, la dimensione in byte dell'array interno). Tale valore non è una costante, ma varia in funzione del contenuto della stringa: qualora fosse necessario, il buffer viene ridimensionato automaticamente.
Entrambi i valori appena detti possono essere recuperati mediante apposite funzioni membro:

- **size()**, che ci dice di quanti caratteri è composta la stringa;
- **capacity()**, che restituisce la massima lunghezza della stringa, prima che le venga allocata nuova memoria.

Già con queste due funzioni possiamo fare degli interessanti esperimenti, atti a valutare dei parametri del nostro sistema o del nostro compilatore.

Diamo uno sguardo al seguente codice:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string s;
  char continua = 's';

  do {
        string c;
        cout << "Inserisci: ";
        cin >> c;

        s.append(c);

        float tot = ((float)s.size()/(float)s.capacity())*100;
        cout << "Occupazione: ";
        cout << s.size() << " su " << s.capacity();
        cout << "\t[ " << tot << "% ]" << endl;
        cout << "Continuiamo? (s/n) ";
        cin >> continua;
        }
        while (continua!='n' && continua!='N');

  return 0;
}
```

Con questo programma, si intende proprio verificare la fluttuazione delle dimensioni del buffer associato ad una stringa in funzione del suo contenuto. È interessante osservare come la dimensione cambi all'improvviso, non appena si superi una certa soglia di riempimento. Nel precedente programma si è fatto uso del casting esplicito (che abbiamo visto nelle prime puntate del corso) per il calcolo della variabile *tot*: questo è stato necessario per via del fatto che le funzioni *size()* e *capacity()* restituiscono un intero, e la divisione tra *s.size()* ed *s.capacity()*, siccome è sempre vero che *s.size()<s.capacity()*, avrebbe sempre dato come risultato 0 (per via del troncamento delle cifre decimali). Per la gestione delle dimensioni di una stringa, oltre alle due appena esposte, esiste anche la funzione *reserve(int)*. Essa ci permette di specificare la dimensione minima del buffer associato ad una stringa, mediante la specifica del parametro passatole. Risulta di utile applicazione qualora fossimo sicuri che lo spazio allocato per la stringa sia in ogni caso insufficiente, oppure se si deve avere una stringa di una certa dimensione prefissata e nota: in questo modo si evita il sovraccarico di lavoro dato dal ridimensionamento del buffer.

MANIPOLAZIONI

Nel precedente programma abbiamo introdotto una nuova funzione, la *append(string)*. Questa funzione esegue un'operazione abbastanza ovvia, come ci suggerisce il suo nome: prende la stringa su cui è invocata e le concatena la stringa passata come parametro. Se vogliamo, è una specializzazione della concatenazione tra stringhe.
Un'altra utile funzione è la *insert(int,string)*. Essa inserisce nella stringa su cui è invocata, alla posizione specificata, la stringa passatale per parametro (si noti che le posizioni si contano a partire da 0). Chiariamo con un esempio:

```
string s1 = "Qui";
string s2 = s1;
s2.append("Qua");
string s3 = s2;
s3.insert(s1.size(),"Quo");
```

La stampa sarà:

Qui Quo Qua

Si presti attenzione, tra l'altro, all'uso che si è fatto della funzione *size()* in questo caso.
Il principale vantaggio della funzione *insert()* è indubbiamente quello di evitare di sovrascrivere parti della stringa originale, evitandoci quindi di dover controllare la cosa per conto nostro.
Un altro punto da notare è il fatto che sia la funzione *append()* che la funzione *insert()* fanno side-effect (cioè, come i lettori più attenti ricorderanno, modificano il loro oggetto di invocazione), ed è per questo che non si è scritto ad esempio:

```
/*   PERICOLO!   */
s3 = s2.insert(s1.size(),"");
/*   PERICOLO!   */
```

Il codice precedente non è sbagliato, si noti, ma probabilmente non fa ciò che deve fare: al termine dell'esecuzione della precedente istruzione, *s2* ed *s3* avranno contenuto identico ma, oltre ad aver creato *s3* secondo i nostri desideri, avremo modificato il contenuto di *s2*, che in situazioni analoghe non è un obiettivo perseguito volontariamente o consapevolmente.
Molto simile alla *insert()* è la funzione *replace (int, int, string)*, che sostituisce una porzione (specificata dai due parametri interi, che indicano il punto da cui iniziare la sostituzione e il numero di caratteri da sostituire) della stringa su cui è invocata con la stringa passatale come parametro. A ben vedere, il lavoro compiuto da questa funzione per conto nostro è notevole: non solo sostituisce una porzione di stringa, ma se necessario (qualora la stringa da inserire abbia lunghezza maggiore della porzione da sostituire) ridimensiona anche la stringa, effettuando gli opportuni sfasamenti atti ad evitare sovrascritture. Vediamo anche qui un esempio:

Manipolazioni in C

Se volete invece dare un'occhiata a una reference delle funzioni standard del C per la manipolazione di stringhe (o meglio: di array di caratteri terminati da "\0" :-) consultate l'indirizzo:

http://www.cplusplus.com/ref/cstring/

```
string s = "Carlo butta la pasta, che e' ora!";
s.replace(15,5,"macchina");
```

Come nei casi precedenti, si nota come anche qui la funzione *replace()* comporta side-effect.
Un'altra funzione utile, che ovviamente (come sarà chiaro tra breve) fa side-effect, è la funzione *erase(int, int)*. Questa funzione si occupa di eliminare un certo numero di caratteri dalla stringa su cui è invocata, a partire dalla posizione specificata dal suo primo parametro. Al solito, facciamo un esempio:

```
string t = "Carlo, che brutta macchina che hai!";
cout << "prima:\t" << t << endl;
t.erase(11,7);
cout << "dopo:\t" << t << endl;
```

Un'ultima nota relativamente alle firme delle funzioni per manipolare stringhe che abbiamo appena visto: di esse esistono diversi overload, e quelle viste sono solo le forme più comode.

RICERCA ALL'INTERNO DI STRINGHE

La classe *string* contiene al suo interno, oltre alle ridefinizioni dei principali operatori e alle funzioni esposte nel precedente paragrafo, anche una serie di funzioni il cui scopo è quello di permetterci di ritrovare, all'interno di una data stringa, le occorrenze di un'altra sottostringa o di un certo carattere. Queste funzioni fanno parte del gruppo di funzioni chiamato, in maniera per nulla sorprendente, *find*.
Le funzioni del gruppo *find* sono riassunte in tabella, insieme a una breve descrizione del loro significato:

Nome	Significato
find()	Ricerca un dato carattere (o gruppo di caratteri) all'interno di una stringa
rfind()	Come find(), ma la ricerca parte dalla fine della stringa e procede a ritroso
find_first_of()	Ricerca in una stringa la prima occorrenza di uno dei caratteri passati come argomento
find_first_not_of()	Ricerca in una stringa la prima occorrenza di un carattere che NON è presente tra quelli passati come argomento
find_last_of()	Ricerca in una stringa l'ultima occorrenza di uno dei caratteri passati come argomento
find_last_not_of()	Ricerca in una stringa l'ultima occorrenza di un carattere che NON è presente tra quelli passati come argomento

Il valore restituito da queste funzioni rappresenta la posizione del primo carattere della stringa cercata. Ovviamente il conteggio (come ovunque in C++) inizia dalla posizione 0. Qualora la ricerca non vada a buon fine viene restituito il valore speciale npos che altro non è che la costante intera -1, accessibile, mediante operatore di scope resolution "::" dalla stessa classe *string*. Un esempio molto semplice è riportato di seguito:

```
//funzione ausiliaria
void cerca(char c, string s)
{
  int pos;
  if ((pos = s.find(c)) == string::npos)
     cout << "Il carattere " << c << " non è presente
                                          nella stringa\n";
  else
     cout << "Il carattere " << c << " si trova in
                               posizione " << pos << endl;
}

//... all'interno del codice ...
string s_ricerca = "questa è una stringa per la ricerca";
cerca('s',s_ricerca);
cerca('z',s_ricerca);
```

questo codice produce, come prevedibile, il seguente output:

Il carattere s si trova in posizione 3
Il carattere z non è presente nella stringa

Un altro modo, molto diffuso, di utilizzare *find()* è quello di adoperare la sua versione a due argomenti, nella quale uno degli argomenti, di tipo *int*, specifica la posizione dalla quale partire per cominciare la ricerca. Questo si rivela molto utile quando si utilizza *find()* all'interno di un ciclo, in quanto permette di analizzare singolarmente ogni occorrenza del testo ricercato. Un esempio di utilizzo di *find()* a due argomenti all'interno di un ciclo è riportato di seguito:

```
string s_ricerca = "questa è una stringa per la ricerca";
string da_ricercare = "er";
int pos = 0;
while ((pos = s_ricerca.find(da_ricercare,pos)) !=
                                          string::npos) {
  cout << "Trovato \"" << da_ricercare << "\"
                          in posizione " << pos << endl;
  pos++;
}
```

L'esempio stampa:

Trovato "er" in posizione 22
Trovato "er" in posizione 31

Come si può vedere abbiamo facilmente ciclato tra le varie occorrenze della stringa *"er"* presenti all'interno della stringa più grande, senza per que-

Sul Web

A chi volesse approfondire la sua conoscenza sulle librerie standard del C++, consigliamo il validissimo libro "Thinking in C++ - 2nd ed. – Volume 2" di Bruce Eckel e Chuck Allison, che rappresenta sicuramente un ottimo riferimento per i programmatori più avanzati (o aspiranti tali) ed è oltretutto disponibile gratuitamente per il download, partendo dall'indirizzo

http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html

sto dovere compiere strane operazioni di Copia&Incolla sulla stringa originale, che infatti non è stata toccata per nulla. Questo, oltre ad essere auspicabile da un punto di vista concettuale (una ricerca non è un'operazione di modifica...), è anche molto utile dal punto di vista pratico in quanto, come d'altra parte avviene per quasi tutte le funzioni standard del C++, l'intera operazione viene effettuata con estrema efficienza e non spreca tempo di calcolo prezioso.
L'altra funzione del tutto analoga a find() che possiamo trovare in questo gruppo è la *rfind()* (= *reverse find*, ricerca all'inverso), che ha esattamente lo stesso comportamento, ad eccezione del fatto che la ricerca parte dalla fine della stringa e procede all'indietro, per cui la prima posizione restituita rappresenta l'ultima occorrenza del testo che si sta cercando. Magari questa funzione non sarà molto utilizzata ma è bene sapere della sua esistenza in quanto, qualora dovesse tornarci utile, ci risparmieremmo una implementazione "al volo" che sicuramente risulterebbe meno efficiente.

TUTTO CIÒ CHE OCCORRE E CHE NON OCCORRE

Le altre quattro funzioni presentate nella tabella, svolgono delle ricerche abbastanza particolari, ma molto più utile di quanto si possa credere: trovano infatti la prima (o l'ultima) occorrenza in una stringa, di un carattere presente nel gruppo di caratteri specificato. Tale gruppo di caratteri può anche essere specificato "al negativo" cioè facendo trovare il primo (o ultimo) carattere che non compare nella lista data.
Come al solito con un esempio chiariremo molte cose:

```
string s_occorrenze = "stringa per la ricerca di occorrenze";
string s_caratteri = "abcd";

//find_first_of()
int pos = s_occorrenze.find_first_of(s_caratteri);
cout << "Il primo carattere di " << s_caratteri <<
                                         " presente è "
      << s_occorrenze[pos] << " in posizione " << pos
                                              << endl;

//find_last_of()
pos = s_occorrenze.find_last_of(s_caratteri);
cout << "L'ultimo carattere di " << s_caratteri <<
                                         " presente è "
   << s_occorrenze[pos] << " in posizione " << pos
                                              << endl;

//find_first_not_of()
pos = s_occorrenze.find_first_not_of(s_caratteri);
cout << "Il primo carattere non presente in "
                               << s_caratteri << " è "
   << s_occorrenze[pos] << " in posizione " << pos
                                              << endl;

//find_last_not_of()
pos = s_occorrenze.find_last_not_of(s_caratteri);
cout << "L'ultimo carattere non presente in "
                               << s_caratteri << " è "
   << s_occorrenze[pos] << " in posizione " << pos
                                              << endl;
```

L'output prodotto è:

Il primo carattere di abcd presente è a in posizione 6
L'ultimo carattere di abcd presente è c in posizione 28
Il primo carattere non presente in abcd è s in posizione 0
L'ultimo carattere non presente in abcd è e in posizione 35

E infatti, come si può ricavare a mente, ad esempio per la prima funzione, il primo carattere del gruppo *"abcd"* è proprio la *"a"* che è presente in posizione 6 nella nostra stringa di prova.
Queste funzioni sono utilissime quando ad esempio si sta implementando un parser, cioè un analizzatore automatico di testo. Per un parser è importante che l'input sia costituito da *"token"* ovvero da singole parole non comprendenti ad esempio spazi, caratteri di tabulazione, caratteri di "a capo" ecc. Con l'uso delle funzioni *find_first_not_of()* e *find_last_ not_of()* si può facilmente ricavare un token da una stringa contenente caratteri "spuri" semplicemente passando come argomento della funzione la stringa:

```
" \a\b\f\n\r\t\v"
```

che appunto contiene, oltre allo spazio iniziale, i vari caratteri speciali non stampabili a schermo.

CONCLUSIONI

Con questa lezione abbiamo concluso la nostra rapida panoramica sul mondo delle stringhe nella libreria standard del C++. Le funzioni messe a disposizione per la manipolazione di questo particolare oggetto informatico sono molteplici e per trattarle tutte servirebbe un manuale apposito (che naturalmente vi invitiamo a trovare e consultare!). Nonostante tutto, il compendio che ne abbiamo dato in queste lezioni, sarà sicuramente molto utile, soprattutto per chi prima d'ora non era neppure a conoscenza dell'esistenza di una classe C++ chiamata *"string"*.
Vi invitiamo dunque a sperimentare l'utilizzo di queste funzionalità anche per prendere confidenza con la libreria standard del C++, in attesa che il nostro viaggio continui.
Alla prossima!

Alfredo Marroccelli e Marco Del Gobbo

Contatta gli autori!

Se hai suggerimenti, critiche, dubbi o perplessità sugli argomenti trattati e vuoi proporle agli autori puoi scrivere agli indirizzi:

alfredo.marroccelli@ioprogrammo.it

e

marco.delgobbo@ioprogrammo.it

Questo contribuirà sicuramente a migliorare il lavoro di stesura delle prossime puntate.

MATLAB

Interazione tra galassie.

Modelli matematici

Nello scorso numero abbiamo imparato a scrivere e ad usare un semplice modello matematico che ci consentisse di riprodurre virtualmente un fenomeno naturale a tutti noto come quello dell'orbita della Terra intorno al Sole e della Luna attorno alla Terra. Abbiamo usato il modello per comprendere come i corpi celesti interagiscono tra loro e quali caratteristiche possono assumere le loro orbite al variare di alcuni parametri specifici.

I modelli, per loro natura, si prestano a cambiamenti e ad evoluzioni che li conducono a descrivere i fenomeni in maniera sempre più fedele: abbiamo avuto modo di vedere che estendere il modello a tre corpi é stato abbastanza semplice. In questo numero tratteremo un numero molto grande di corpi e cercheremo di comprendere quali siano le interazioni gravitazionali tra oggetti celesti molto più complessi del nostro sistema solare, due galassie. Questo lavoro prende spunto dai famosi lavori che trovate citati nella nota bibliografica al fondo dell'articolo.

IL PERCHÉ DI UNA FORMA

Tutti questi autori hanno tentato di spiegare la forma delle attuali galassie per mezzo di modelli (i più "antichi" implementati in FORTRAN mentre i più recenti in Basic) che spiegassero come mai le galassie odierne possiedano tali forme (come é possibile vedere in Fig. 1). Essi ipotizzarono che la forma fosse conseguenza di una pura interazione gravitazionale o che comunque questa fosse la causa principale di così peculiari deformazioni. Per replicare il loro lavoro, adotteremo molte delle considerazioni originali tratte da tali scritti, in modo da implementare in MATLAB un modello equivalente. Se pensiamo a quanto complesso sia un oggetto come una galassia potremmo rapidamente decidere che un buon modello é impossibile da implementare, sia per la complessità delle interazioni tra i singoli corpi sia per il loro grande numero. Se non ci lasciamo scoraggiare possiamo scoprire con un certo sollievo che alcune di queste complessità possono essere superate. Adotteremo una terminologia, richiamata nel numero precedente, che si presta bene a descrivere questo caso: chiamiamo il corpo celeste che ha maggiore massa e che risiede inizialmente più vicino al centro di rotazione del sistema (centro di massa), *"bersaglio"*, mentre il secondo lo chiameremo *"intruso"*. Considereremo l'intruso come il corpo che con il suo campo gravitazionale va a perturbare la quiete o il moto uniforme del bersaglio. Il sistema è ora però molto più complesso di quello Sole-Terra poiché le galassie possiedono una forma ed una distribuzione delle stelle assolutamente non banale. Siamo quindi costretti a semplificare il nostro mo-

Fig. 1: Immagine di M51 (NGC5194/5).

dello di pensiero per rendere il modello algoritmico più trattabile.

IPOTESI

Supponiamo che il sistema *Bersaglio/Intruso* sia chiuso (non ha interazioni di sorta con altri oggetti celesti); questo ci consente di adottare un modello ristretto all'interazione tra due sole galassie. Stabiliamo che le stelle della galassia siano distribuite su un disco piatto attorno al nucleo; non modelleremo, quindi, le stelle presenti nell'alone così come gli ammassi globulari che solitamente stanno al di fuori del piano galattico. Stabiliamo che le masse delle galassie sono concentrate nel loro nucleo e che le stelle che formano il disco non influenzano i nuclei con la loro massa e che non subiscono l'influenza delle altre stelle a loro vicine; in questo modo possiamo evitare di calcolare le migliaia di interazioni generate da tali stelle. Nei lavori originali citati si assumeva anche che il *Bersaglio* fosse fermo all'inizio della simulazione e che l'*Intruso* non possedesse stelle. Vedremo, nel seguito, che possiamo sbarazzarci di queste due limitazioni poiché possediamo il nostro modello che ci permette di calcolare lo spostamento del *Bersaglio* e se riusciremo a calcolare gli spostamenti delle stelle del *Bersaglio* non vi é motivo per non riuscire a fare lo stesso per quelle dell'*Intruso*.
In effetti, quest'ultima considerazione era in origine stata fatta per motivi legati alla velocità di calcolo dei sistemi allora a disposizione.
Rispetto al modello Sole-Terra dobbiamo immaginarci alcune modifiche che ci permettano di includere nel modello le stelle del *Bersaglio* e le stelle dell'*Intruso*. Ovviamente dobbiamo anche immaginare di fornire una forma e una velocità iniziale appropriata ai dischi galattici. Spendiamo ancora due parole per vedere come rendere il modello compatibile con la scala del problema che ci accingiamo a risolvere. Mentre nel numero scorso potevamo ancora avvalerci del sistema MKS *(metri, kilogrammi, secondi),* in questo caso la scala galattica del problema e l'uso di tali unità di misura ci costringerebbero ad adottare numeri poco maneggevoli e di magnitudine enorme. Invece, qui adottiamo unità di misura più appropriate al contesto. Se diamo un'occhiata allo script di lancio del modello (*simulgx.m*) notiamo subito due cose: il numero di parametri é aumentato a dismisura e le unità di misura che abbiamo scelto di usare ci consentono l'uso di numeri molto comodi e leggibili. Concentriamoci su alcune di queste variabili per comprendere bene quali siano le unità ora utilizzate. Per esempio, la massa delle galassie viene espressa in miliardi di masse solari. La massa del *Bersaglio* viene ad essere pari a 100 miliardi di masse solari (la Via Lattea ha una massa pari a circa 200 miliardi di masse solari). Diamo ora un'occhiata alle posizioni: queste sono espresse in kiloparsec. Un parsec é la distanza alla quale si osserva la traiettoria della Terra intorno al Sole (la sua orbita) sotto un angolo di un secondo d'arco (da qui il nome di *parallasse-secondo, parsec*) ed é pari a 3.262 anni luce oppure 3.086e16 metri e quindi *1 kpc= 3.086e19 m.* Mentre le velocità sono espresse come *kpc al milione di anni.* Tutti gli altri parametri presenti in *"simulgx.m"* saranno più chiari non appena procederemo ad esplorare il modello di calcolo. Prima di farlo dichiariamo lo scopo del nostro esperimento. Nei lavori citati, specialmente quello dei fratelli *Toomre*, si ipotizza che la forma delle attuali galassie sia dovuta ad interazioni gravitazionali tra galassie.
Ovviamente, un' assunzione di questo tipo non é possibile verificarla poiché é alquanto poco pratico registrare un alto numero di osservazioni in un intervallo di tempo significativo (milioni di anni). Una soluzione ad un tale problema é possibile trovarla per mezzo della simulazione e quindi della realizzazione e sviluppo di un appropriato modello della realtà. In altre parole, desideriamo vedere se é vero che una pura interazione gravitazionale é in grado (pur con tutte le approssimazioni alle quali siamo costretti dalla complessità del fenomeno reale) di deformare una galassia sino a farla assomigliare a quelle che l'uomo osserva quotidianamente con i suoi strumenti scientifici.

IL MODELLO ALGORITMICO

Immergiamoci finalmente nel nostro modello algoritmico ed esploriamolo per comprendere le implicazioni implementative celate in esso. Ci limiteremo a commentare soltanto quelle parti che suscitano un certo interesse dal punto di vista del modello oppure dell'uso di MATLAB in problemi di questo tipo. Il modello risiede nel programma *galaxy.m*.

```
% Inizializzazione variabili
% G = 6.67259 10^(-11) N * m^2 / kg^2
G = .44995*10^(-2);  % kpc^3 / (Mal^2 * GigaMo),
        dove Mo = Masse solari e Mal = Milioni di anni luce
```

Il valore originale di *G* (costante di *gravitazione universale*) deve essere modificato, poiché ora utilizziamo un sistema di misura differente dallo MKS. Come accennato sopra, non usiamo più i metri ma i *kpc*, non più i kilogrammi ma i miliardi di masse solari, ecc.
Effettuando le opportune conversioni troviamo il nuovo valore che ci consente di continuare a calcolare le forze gravitazionali in maniera corretta.

[1]

```
dr = radius_tgt / (anelli_tgt - 1);
nrs = stellePrimoAnello_tgt;

% Stelle della galassia TARGET
i = 0;
TGT_pos=[];
TGT_vel=[];
```

[1] **L'espressione [] denota l'array vuoto e serve ad inizializzare una variabile che mostrera' una dimensione di zero righe e zero colonne**

MATLAB

[2] L'apice usato in corrispondenza di una parentesi quadra induce il calcolo della trasposta (un vettore riga diviene colonna e viceversa)

[3] Vengono imposte tre rotazioni successive di posizioni e velocita': la prima attorno all'asse X, la seconda attorno all'asse Y e la terza attorno all'asse Z

[4] Le matrici *TGT_pos* e *TGT_vel* contengono le componenti *x, y* e *z* (colonne) di posizioni e velocita' delle stelle del bersaglio

```
for iesimo_anello=1:anelli_tgt
    r = radius_tgt * .2 + (iesimo_anello - 1) * dr;
    V = sqrt(G * massa_tgt / r);
    nrs = nrs + round((iesimo_anello>=1) * nrs / 20);
    [righeT,colonneT] = size(TGT_pos);
    i = [righeT + 1 : 1 : righeT + nrs];
    % Posizioni iniziali
    t = [0 : 2 * pi / nrs : 2 * pi];
    t = t(1 : end-1);
    TGT_pos(i,1) = [r .* cos(t)]' + x_tgt;
    TGT_pos(i,2) = [r .* sin(t)]' + y_tgt;
    TGT_pos(i,3) = z_tgt;
    % Velocita' iniziali
    if moto_tgt
        segno = -1;
    else
        segno = +1;
    end
[2] TGT_vel(i,1) = [-segno * V .* sin(t)]' + vx_tgt;
    TGT_vel(i,2) = [segno * V .* cos(t)]' + vy_tgt;
    TGT_vel(i,3) = vz_tgt;
end
```

Questa porzione di codice é stata scritta con lo scopo di posizionare le stelle su un disco attorno al nucleo e di determinare la loro velocità di rotazione. Prima di tutto viene determinata una distanza tra gli anelli che formano il disco (*dr*). Quindi viene usato il numero di stelle che risiedono sul primo anello come dato di partenza (*nrs*).
Da notare che non viene specificato un numero totale di stelle ma questo viene invece determinato a partire dal numero di anelli che si vogliono usare per la rappresentazione del disco e da quante stelle risiedono sul primo anello. Non appena entriamo nel ciclo (anello per anello) calcoliamo il raggio (*r*) al quale si trova l'anello in questione, quindi il modulo della velocità che un corpo celeste deve avere a quella distanza dal nucleo della galassia (con massa concentrata nel nucleo). La successiva linea di codice é di un qualche interesse dal punto di vista della sintassi delle espressioni matematiche in MATLAB. Concentriamoci sulla seconda parte *(round((iesimo_anello>=1) * nrs / 20))*; è curioso trovarvi un'espressione logica come *iesimo_anello>=1*. Ma solo fino ad un certo punto: se riflettiamo bene si tratta di un'espressione che può restituire uno *0* (nel caso in cui si stia processando il primo anello) o un *1* (tutti gli altri anelli). Nel primo caso, l'intera seconda parte del calcolo assume il valore *0*, mentre, nel secondo caso, viene aggiunto a *nrs* un numero di stelle pari a *nrs/20*. Ne consegue che il primo anello ha un numero di stelle pari a *nrs*, mentre i successivi hanno un numero di stelle sempre crescente con un tasso pari ad un ventesimo dell'anello precedente.
Il codice che segue calcola le dimensioni di *TGT_pos* (viene costruita come una matrice di *n* righe e 3 colonne; le colonne contengono le coordinate *x*, *y* e *z* delle stelle della galassia bersaglio). La riga di codice successiva calcola l'intervallo di indici di riga che verranno utilizzati in seguito per memorizzare le posizioni delle stelle dell'anello in questione; ne abbiamo bisogno perché la grossa matrice *TGT_pos* contiene sequenzialmente tutte le stelle. Quindi calcoliamo le equazioni parametriche del cerchio che ha come centro il nucleo della galassia e come raggio quello dell'anello in questione. La variabile "*moto_tgt*" impone un moto normale o retrogrado delle stelle della galassia bersaglio. E finalmente vengono calcolate le velocità nelle loro componenti lungo gli assi *x*,*y* e *z* e vengono poste, a loro volta, in una matrice di dimensioni identiche a quella che memorizza le posizioni. La matrice "*TGT_vel*" possiede anche essa *n* righe, tante quante sono le stelle della galassia e tre colonne contenenti le tre componenti del vettore velocità. In "*galaxy.m*" troviamo subito dopo il codice che esegue lo stesso identico compito per le stelle della galassia intruso:

```
% Se le due galassie non sono sullo stesso piano allora
% consideriamo l'intrusore come ruotato (il target funge
                                          da riferimento)
[righe,colonne] = size(INT_pos);
for i=1:righe
    % Rotazione delle posizioni
[3] [INT_pos(i,1),INT_pos(i,2),INT_pos(i,3)]=
        rotaz(INT_pos(i,:),[0 0],xrot,[x_int y_int z_int]);
    [INT_pos(i,1),INT_pos(i,2),INT_pos(i,3)]=
        rotaz(INT_pos(i,:),[0 90],zrot,[x_int y_int z_int]);
    [INT_pos(i,1),INT_pos(i,2),INT_pos(i,3)]=
        rotaz(INT_pos(i,:),[90 0],yrot,[x_int y_int z_int]);
    % Rotazione delle velocita'
    [INT_vel(i,1),INT_vel(i,2),INT_vel(i,3)]=
        rotaz(INT_vel(i,:),[0 0],xrot,[vx_int vy_int vz_int]);
    [INT_vel(i,1),INT_vel(i,2),INT_vel(i,3)]=
        rotaz(INT_vel(i,:),[0 90],zrot,[vx_int vy_int vz_int]);
    [INT_vel(i,1),INT_vel(i,2),INT_vel(i,3)]=
        rotaz(INT_vel(i,:),[90 0],yrot,[vx_int vy_int vz_int]);
end
```

Questa porzione di programma fa ruotare la galassia intruso di un ammontare determinato dalle variabili *xrot*, *yrot* e *zrot*. La routine di rotazione lavora in maniera differente da quella presentata in uno degli scorsi numeri. Quella utilizzata qui è più sofisticata poiché consente la rotazione attorno ad un asse qualunque specificato per mezzo di azimuth ed elevazione. Le rotazioni attorno ad assi predeterminati ci consente di considerare galassie che non giacciano su piani paralleli e, quindi, ci permette di sperimentare situazioni più realistiche e complesse. Passiamo ora al cuore del modello di calcolo. Citiamo qui solo alcuni passaggi in modo da renderne più chiara la trattazione:

[4]

```
% Calcolo delle velocita' delle stelle della galassia TARGET
opt1 = G*massa_tgt;
opt2 = G*massa_int;
```

```
% Il termine +1 kpc nell'espressione e' il fattore di abbassamento
% della forza di attrazione mano a mano che ci si avvicina al
% nucleo. Questo per evitare che vi siano forze troppo grandi
% all'approssimarsi del centro di massa di una delle due galassie.
opt3=((TGT_pos(:,1)-x_tgt).^2 + (TGT_pos(:,2)-y_tgt).^2 + (TGT_pos(:,3)-z_tgt).^2 + 1).^1.5;   [5]
opt4=((TGT_pos(:,1)-x_int).^2 + (TGT_pos(:,2)-y_int).^2 + (TGT_pos(:,3)-z_int).^2 + 1).^1.5;
TGT_vel(:,1) = TGT_vel(:,1) + opt1./opt3.*(x_tgt-TGT_pos(:,1)) + opt2./opt4.*(x_int-TGT_pos(:,1));
TGT_vel(:,2) = TGT_vel(:,2) + opt1./opt3.*(y_tgt-TGT_pos(:,2)) + opt2./opt4.*(y_int-TGT_pos(:,2));
TGT_vel(:,3) = TGT_vel(:,3) + opt1./opt3.*(z_tgt-TGT_pos(:,3)) + opt2./opt4.*(z_int-TGT_pos(:,3));
% Calcolo della nuova posizione delle stelle della galassia TARGET
TGT_pos = TGT_pos + TGT_vel;
% Calcolo delle velocita' delle stelle della galassia INTRUDER
. . . (omissis) . . .
% Calcolo della velocita' del centro della galassia TARGET
opt1 = x_tgt - x_int;
opt2 = y_tgt - y_int;
opt3 = z_tgt - z_int;
opt4 = (opt1^2 + opt2^2 + opt3^2 + 1)^1.5;
vx_tgt = vx_tgt - G*massa_int*opt1/opt4;
vy_tgt = vy_tgt - G*massa_int*opt2/opt4;
vz_tgt = vz_tgt - G*massa_int*opt3/opt4;
% Calcolo della velocita' del centro della galassia INTRUDER
. . . (omissis) . . .
% Aggiornamento della posizione del centro della galassia TARGET
x_tgt = x_tgt+vx_tgt;
y_tgt = y_tgt+vy_tgt;
z_tgt = z_tgt+vz_tgt;
% Aggiornamento della posizione del centro della galassia INTRUDER
. . . (omissis) . . .
```

Il modello é molto simile a quello visto la volta precedente per calcolare l'orbita della Terra attorno al Sole. Qui troviamo anche il calcolo dell'influsso che i nuclei galattici inducono sulle stelle appartenenti ad entrambe le galassie e, in accordo con le semplificazioni illustrate sopra, non troviamo il calcolo delle forze generate dalle singole stelle. In molti casi vengono usate variabili chiamate *opt*… ma questo viene fatto soltanto per comodità e velocità di calcolo. Notiamo che qui non troviamo più il termine relativo al tempo trascorso (*delta t*).
Questo avviene perché consideriamo l'intervallo di tempo più opportuno quello relativo ad un milione di anni. Ovviamente, per chi volesse sperimentare differenti passi temporali, si dovrà modificare il codice nel calcolo delle velocità e delle posizioni in accordo con il modello visto nel numero precedente.

SIMULAZIONE

Ora possediamo il modello e non ci resta che iniziare la nostra fase di sperimentazione. Iniziamo con un insieme di dati di base che sono contenuti nello script di lancio del modello (*simulgx.m* oppure *m51.m*) e che sono quelli che potete trovare nella letteratura citata. Essi sono tesi a riprodurre un'interazione simile a quella che conduce ad una configurazione di galassie simile a quella di *M51* (Fig. 1).

Fig. 2: **Risultato della simulazione (vista in prospettiva).**

Il risultato della simulazione lo vediamo invece in Fig. 2 (in prospettiva) ed in Fig. 3 (in pianta).

Fig. 3: **Risultato della simulazione (vista in pianta).**

I nuclei galattici sono rappresentati in nero, le traiettorie dei nuclei in giallo, le stelle del bersaglio in celeste e quelle dell'intruso in rosso. Vale qui la pena di focalizzarsi su un aspetto peculiare della routine implementata per calcolare posizioni e velocità delle stelle. Possiamo facilmente notare che tutto il codice è stato vettorizzato. La vettorizzazione è una tecnica che porta a strutturare i dati in modo da po-

MATLAB

[5]
Il carattere ":" in questo caso indica tutte le righe di una matrice

Sul Web

Getting Started with MATLAB
http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf

Using MATLAB
http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/using_ml.pdf

Using MATLAB Graphics
http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/graphg.pdf

Galactic Dynamics
http://www.astro.utu.fi/~cflynn/galdyn/

MATLAB

ter applicare le leggi dell'algebra lineare alla risoluzione di problemi di calcolo; il codice che ne risulta é particolarmente conciso.

Fig. 4: NGC 4038/4039.

In altri linguaggi saremmo stati costretti a scandire ogni elemento delle matrici di posizione e velocità e ad eseguire il calcolo stella per stella. Qui, invece, abbiamo organizzato i dati in maniera che fosse possibile applicare un'espressione matematica a tutto un insieme ed ottenere così il risultato di un calcolo con una sola operazione e senza il ricorso alla scrittura di cicli *for*. Per terminare, ci concentriamo sulla riproduzione di esempi notevoli presi dalla realtà per poter così ottenere il risultato che ci eravamo prefissi all'inizio.
Se dobbiamo suggerire che le forme delle attuali galassie siano probabilmente derivate da interazioni gravitazionali tra loro dobbiamo poter fornire una casistica più ampia. Ci proviamo per mezzo di un esempio notevole, quello di *NGC 4038/4039*, più nota come *"Antennae"*.
Il codice seguente carica l'immagine delle galassie in questione e ne visualizza la corrispondente immagine:

[6]

```
>> [im,map]=imread('antennae.bmp');
>> image(im)
>> colormap(map)
>> axis off
>> title ('NGC 4038 / 4039')
>> axis equal
```

Il risultato é visibile in Fig. 4.
Nello script *antennae.m* é invece possibile trovare l'insieme di parametri che inducono una simulazione a comportarsi in una maniera simile a quella delle due galassie reali e produce il risultato di Fig. 5.

Fig. 5: Simulazione di NGC 4038/4039.

CONCLUSIONI

In questo numero abbiamo visto un aspetto molto importante della trattazione dei problemi per mezzo di modelli: abbiamo dovuto prendere delle decisioni importanti circa la complessità del modello da implementare e siamo così stati in grado di riprodurre, in maniera virtuale, situazioni altrimenti non sperimentabili nella realtà. La susseguente sessione di test e tentativi ci ha insegnato anche alcune proprietà del nostro modello che non erano subito così evidenti: per esempio, notiamo che la deformazione dei dischi galattici dipende dalla massa delle galassie ma anche, in misura significativa, dal tempo durante il quale una galassia rimane esposta al campo gravitazionale di un'altra. Inoltre, tutto questo stimola ulteriori evoluzioni del modello; una delle più banali, ma che può rivelarsi utile, potrebbe essere quella che ci consente di conservare la forma di una galassia perturbata dal passaggio di un intruso per farle sperimentare il passaggio di un nuovo intruso che la perturba una seconda volta. Ancora, sarebbe estremamente istruttivo comprendere quale sia la deformazione di altri due componenti della galassia: il rigonfiamento centrale attorno al nucleo e l'alone. Il rigonfiamento centrale é un insieme di stelle a stretto contatto con il nucleo che ha una distribuzione approssimativamente sferica delle stelle. L'alone invece é formato da stelle e ammassi globulari che risiedono attorno al nucleo della galassia in uno spazio approssimativamente sferico ad una distanza massima pari al raggio del disco ma che non giacciono sul disco stesso.
Come sempre, invito tutti coloro i quali vogliano contribuire con suggerimenti e idee a contattarmi direttamente per mezzo dell'indirizzo di posta elettronica citato sotto.
Per maggiori informazioni sui prodotti della famiglia MATLAB potete consultare il sito di The MathWorks (*www.mathworks.it*). Suggerisco a tutti di dare un'occhiata ad una porzione del sito web chiamato "MATLAB Central" (*www.mathworks.com/matlabcentral/*) . Esso riporta innumerevoli esempi di uso di MATLAB in una varietà molto vasta di discipline tecnico scientifiche.

Fabrizio Sara

[6]
La funzione "imread" legge immagini grafiche nei formati più noti e genera due matrici nel workspace (valori e mappa di colore)

Bibliografia

• **GALACTIC BRIDGES AND TAILS**
Alar e Juri Toomre
(The Astrophysical Journal)
1972

• **GALACTIC COLLISIONS ON YOUR COMPUTER**
Michael C. Schroeder e Neil F. Comins
(Astronomy)
Dicembre 1988

• **GALASSIE PLASMATE DAGLI URTI**
Fabio Falchi
(L'Astronomia numero 144)
Giugno 1994

Utilizzare le API di sistema.

Esplosione ed implosione di un form

In questo appuntamento implementeremo un'applicazione che utilizza effetti speciali come esplosione,implosione e trasparenza.

Nel precedente appuntamento abbiamo descritto alcune funzioni, dell'API di Windows, che consentono di migliorare l'aspetto dei Form, in particolare abbiamo visto funzioni della categoria *Region (CreateRectRgn, CreateEllipticRgn, CombineRgn)* e della categoria *Painting* e *Drawing (SetWindowRgn)*; come esempio abbiamo implementato una finestra di forma circolare. Oltre a queste abbiamo descritto alcune funzioni per il controllo dei movimenti del *Mouse (SendMessage e ReleaseCapture)* e per la gestione della memoria *(DeleteObject)*. Infine, abbiamo introdotto l'applicazione *"Visualizzatore Immagini"* che implementeremo in questo appuntamento. Poiché alcune parti di questa applicazione compaiono e scompaiono con l'effetto esplosione/implosione, nel corso dell'articolo presenteremo la sintassi dell'insieme delle funzioni dell'API che consentono di attivare questi effetti; inoltre, parallelamente, miglioreremo la tecnica per rendere un form trasparente.

ESPLOSIONE ED IMPLOSIONE

L'esplosione di un finestra di forma rettangolare può essere simulata disegnando aree rettangolari di estensione crescente. Le aree devono essere tracciate a partire dal punto medio della finestra e devono crescere, proporzionalmente, fino a coprire l'intero form; infine su questi rettangoli deve essere mostrato il form. Per l'implosione si deve procedere al contrario: nascondere il form e poi disegnare tanti rettangoli di superficie decrescente. Quindi, s'intuisce, che abbiamo bisogno di funzioni API che ci permettano di disegnare, riempire, controllare ed eliminare superfici rettangolari.

Prima di descrivere queste funzioni ricordiamo che un rettangolo può essere identificato da una struttura di tipo *Rect*.

```
Type RECT
        Left As Long
        'coordinata X1
        Top As Long
         'Y1
        Right As Long
         'X2
        Bottom As Long
         'Y2
End Type
```

Questa struttura definisce un rettangolo attraverso le coordinate di due punti (*upper left* e *lower-right*) punto superiore sinistro *(left, top)* e punto inferiore destro (right, bottom). Iniziamo a descrivere le funzioni API.

Fig. 1: Il Form FrmTrasparente in fase di progettazione.

LE FUNZIONI API

In generale, per conoscere la superficie del rettangolo che copre l'intero form, possiamo utilizzare la funzione seguente:

```
Declare Function GetWindowRect Lib "user32"
        (ByVal hwnd As Long, lpRect As RECT) As Long
```

Garbage Collection

Utilizzando componenti COM all'interno del framework .NET, si continua a beneficiare dell'intervento del *Garbage Collector.* Come noto, la cancellazione degli oggetti da parte di .NET, avviene su base non determnistica quindi, nel caso in cui i componenti COM utilizzino risorse in modo particolarmente dispendiose, sarà bene prevedere un metodo *Dispose()* che permetta di rilasciare esplicitamente l'oggetto.

Visual Basic

Hwnd è l'handle del form, *LpRect* è un puntatore ad un *Rect* che viene creato in base alle coordinate *(upper-left* e *lower-right)* del form. Per disegnare un rettangolo invece usiamo la seguente:

```
Declare Function Rectangle Lib "gdi32" (ByVal hdc As
        Long, ByVal X1 As Long, ByVal Y1 As Long, ByVal
        X2 As Long, ByVal Y2 As Long) As Long
```

Fig. 2: Il Form FrmTrasparente in fase di esecuzione.

I parametri *X1, Y1, X2, Y2* sono le coordinate dei punti necessari per disegnare un rettangolo; *hdc*, invece, è il *"device context"* che si ricava attraverso la seguente:

```
Declare Function GetDC Lib "user32.dll" (ByVal hWnd
        As Long) As Long
```

hWnd identifica l'area o la finestra di cui si vuole conoscere il contesto. Se il parametro è nullo viene restituito il contesto di tutto lo schermo. Il rettangolo disegnato è riempito con il colore del *Brush* (spazzola) corrente. Il Brush può essere ridefinito attraverso la *CreateSolidBrush*.

```
Declare Function CreateSolidBrush Lib "gdi32" (ByVal
        crColor As Long) As Long
```

In particolare, questa funzione crea un *Brush* con il colore specificato attraverso il parametro *crColor*. Sia il *Device Context* che il *Brush* devono essere rilasciati dopo l'uso, questo viene fatto attraverso le seguenti funzioni:

```
Declare Function ReleaseDC Lib "user32" (ByVal hwnd
        As Long, ByVal hdc As Long) As Long
Declare Function DeleteObject Lib "gdi32" (ByVal
        hObject As Long) As Long
```

ReleaseDC rilascia il contesto mentre *DeleteObject* rilascia l'oggetto *Brush*. Ora possiamo mettere insieme gli elementi descritti e creare la funzione che fa "esplodere" il Form.

```
Sub EspandiForm(frm As Form, passaggi As Long)
`il parametro passaggi imposta la velocità dell'esplosione
Dim colore As Long
Dim FormRect As RECT
Dim FormWidth As Integer
Dim FormHeight As Integer
Dim SD As Long, spazzola As Long
Dim i As Long, X As Integer, Y As Integer
DIM XRect As Integer, YRect As Integer
colore = &HFFC0C0
'si potrebbe usare frm.BackColor
 GetWindowRect frm.hwnd, FormRect
 FormWidth = FormRect.Right - FormRect.Left
 FormHeight = FormRect.Bottom - FormRect.Top
`così sono calcolate le dimensioni del form
 SD = GetDC(0)
 spazzola = CreateSolidBrush(colore)
 For i = 1 To passaggi
   XRect = FormWidth * (i / passaggi)
   YRect = FormHeight * (i / passaggi)
   X = FormRect.Left + (FormWidth - XRect) / 2
   Y = FormRect.Top + (FormHeight - YRect) / 2
  `così sono calcolate le coordinate dei punti del rettangolo
   Rectangle SD, X, Y, X + XRect, Y + YRect
 Next i
 ReleaseDC 0, SD
`rilascia il contesto
 deleteObject (spazzola)
End Sub
```

Notate che *XRect* è compreso tra *FormWidth /passaggi* e *FormWidth* e che *YRect* è compreso tra *FormHeight/passaggi* e *FormHeight* e, di conseguenza, *X* è compreso tra circa *FormRect.Left+ FromWidth/2* e *FormRect.Left* ecc.
Quindi, come accennato, verranno disegnati rettangoli di dimensioni crescenti. Per usare la *EspandiForm* sono necessarie le seguenti istruzioni:

```
EspandiForm Form1, 2500
Form1.Show
```

Queste istruzioni prima disegnano i rettangoli, poi mostrano il *Form1*. Ecco, inoltre, la funzione che gestisce l'implosione:

```
Sub implodiform(frm As Form, passaggi As Long)
   Dim colore As Long
   Dim FormRect As RECT, FormWidth As Integer
   Dim FormHeight As Integer
   Dim SD As Long, spazzola As Long
   Dim i As Long, X As Integer, Y As Integer, XRect As
                                                Integer
 Dim YRect As Integer
 colore = frm.BackColor
 `va bene anche colore = codice colore
 `oppure passare il colore come parametro della funzione
 GetWindowRect frm.hwnd, FormRect
 FormWidth = (FormRect.Right - FormRect.Left)
 FormHeight = FormRect.Bottom - FormRect.Top
 SD = GetDC(0)
```

Visualizzatore Immagini

Il visualizzatore Immagini e Fax di Windows XP è facilmente importabile su un form Visual Basic. La libreria da importare è "Preview 1.0 Type Library".

```
spazzola = CreateSolidBrush(colore)
For i = passaggi To 1 Step -1
  XRect = FormWidth * (i / passaggi)
  YRect = FormHeight * (i / passaggi)
  X = FormRect.Left + (FormWidth - XRect) / 2
  Y = FormRect.Top + (FormHeight - YRect) / 2
  Rectangle SD, X, Y, X + XRect, Y + YRect
Next i
ReleaseDC 0, SD
DeleteObject (spazzola)
End Sub
```

Tra poche righe vedremo dove usare queste funzioni.

- **Il Visualizzatore Immagini -** Il visualizzatore immagini presenta due finestre: la *FrmRicerca,* che permette di ricercare le immagini e la *FrmTrasparente* che permette di sfogliarle. Quest'ultima l'abbiamo nominata "trasparente" perché ha una forma particolare, impostata utilizzando le funzioni API viste nel precedente articolo.

Fig. 3: Il Visualizzatore Immagini di Windows XP inserito in un form.

- **Il form di ricerca-** La *FrmRicerca* è il tipico form per la ricerca di file. Essa consente di esplorare i dischi e le directory del computer, si veda la Fig. 4. La *FrmRicerca* permette di esplorare il File System attraverso un *Dirlistbox* (nominato *drvList*), un *Filelistbox* (nominato *filList*) e un *Drivelistbox* (nominato *DirList*). Il codice per gestire questi elementi è stato presentato nei nostri precedenti articoli per questo non lo illustreremo.

Fig. 4: Il Form di ricerca

- **Il form che gestisce le immagini -** La *FrmTrasparente* gestisce le immagini trovate attraverso la *FrmRicerca.* Sulla *FrmTrasparente* le immagini vengono mostrate attraverso un frame della libreria *MsForms* (in realtà si dovrebbe usare un controllo Activex ad hoc, sicuramente più efficiente). La *FrmTrasparente* presenta una barra di comando (costituita da una serie di *CommandButton*) che permette di ricercare le immagini, di caricarle nel frame, di sfogliarle (con i tasti *avanti* e *indietro*), di proiettarle (con *Bplay*) e d'ingrandirle (con i tasti *Zoom +* e *Zoom -*). Sul form inoltre sono presenti i comandi per fermare la proiezione *(Stop),* per allargare la finestra e per chiudere l'applicazione (si controlli la Fig. 1). Il form di ricerca viene attivato cliccando sul commandbutton *CercaFile* (pulsante con l'icona di esplora risorse). Per quanto riguarda l'interazione tra i due Form, essa deve avvenire con il passaggio di informazioni dalla *FrmRicerca* alla *FrmTrasparente.* In particolare, bisogna passare il nome e il *Path* delle immagini trovate. Questo lo facciamo attraverso la *Form_Unload* della *FrmRicerca* e la procedura *"CaricaPrima"* definita sulla *FrmTrasparente.*

```
Private Sub Form_Unload(Cancel As Integer)
 Dim i As Integer
 If Me.filList <> "" Then
  Dim lista As New Collection
  For i = 0 To filList.ListCount - 1
   lista.Add filList.List(i)
  Next
 FrmTrasparente.caricaprima lista, dirList.path
 Else
  MsgBox "La directory non contiene immagini"
 End If
 ImplodiForm Frmricerca, 2500
End Sub
```

Il codice della *CaricaPrima* lo vedremo a breve. Allora, quando la directory selezionata contiene immagini, e quindi la filelistbox non è vuota, nella *Unload* della *FrmRicerca*, viene creata e caricata la collection *"lista"* che poi, attraverso la funzione "caricaprima", viene passata al form *FrmTrasparente.* S'intuisce che la collection *"lista"* contiene il nome delle immagini e che *dirList .path* rappresenta il path della directory che contiene le immagini. Dopo questi passaggi la *FrmRicerca* viene fatta "implodere". Prima di descrivere le varie parti del form *FrmTrasparente*, ve-

Visual Basic

Style

Per associare una immagine ad un *CommandButton*, della libreria VB, bisogna impostare la proprietà *Style* su *Graphical* e poi specificare l'immagine attraverso la proprietà *Picture*.
Nel nostro esempio ad ogni pulsante abbiamo associato un'immagine.

Visual Basic

diamo le dichiarazioni da inserire nel progetto (queste, naturalmente, possono essere inserite in un modulo *.Bas*). Innanzitutto, bisogna specificare le dichiarazione di tutte le funzioni API introdotte in questo e nel precedente articolo, poi, bisogna specificare le dichiarazioni per la gestione dei parametri passati dal form *FrmRicerca* cioè:

```
Public listafile As Collection
'contiene i valori della collection "lista"
 Public pathdir As String
'contiene il Path della directory con le immagini
Public numerofoto As Integer
'indica la foto mostrata
```

Ora descriviamo, brevemente, le procedure associate ad alcuni pulsanti. Il pulsante *cercafile* mostra la *FrmRicerca* con l'effetto *"esplosione"*.

```
Private Sub cercafile_Click()
 espandiform Frmricerca, 2500
 Frmricerca.Show 1
End Sub
```

Come accennato, nell'*UnLoad* della *FrmRicerca*, per passare i parametri alla *FrmTrasparente*, viene richiamata la *"CaricaPrima":*

```
Public Sub CaricaPrima(lista As Collection, path
                                    As String)
 numerofoto = 0
 Set listafile = lista
 pathdir = path
 avanti_Click
'associata al commandbutton Avanti
End Sub
```

La procedura *avanti_Click* serve per caricare, nel frame (nominato *framefoto*), la prima immagine (quando il frame è vuoto) o l'immagine successiva. Facciamo notare che, ora, i nome delle immagini sono contenute nella collection *"listafile"*.

```
Private Sub avanti_Click()
 numerofoto = numerofoto + 1
 If numerofoto <= listafile.Count Then
  framefoto.Picture = LoadPicture(pathdir & "\"
                         & listafile.Item(numerofoto))
  Else
  numerofoto = numerofoto - 1
 End If
End Sub
```

Invece, per caricare l'immagine precedente, usiamo la *Dietro_Click*, naturalmente associata al pulsante *Dietro*.

```
Private Sub dietro_Click()
 numerofoto = numerofoto - 1
If numerofoto >= 1 Then
 framefoto.Picture = LoadPicture(pathdir & "\"
                         & listafile.Item(numerofoto))
Else
 numerofoto = numerofoto + 1
End If
End Sub
```

Per mostrare le immagini in sequenza utilizziamo del codice inserito in un timer *(timerplay)* attivato e disattivato attraverso i pulsanti *bplay* e *bstop*. In particolare, nel *Timer* e nei pulsanti, prevediamo il seguente codice (naturalmente la velocità di scorrimento delle immagini viene settata attraverso la proprietà *Interval* del *Timer*)

```
Private Sub bplay_Click()
 numerofoto = 0
 Timerplay.Enabled = True
End Sub

Private Sub bstop_Click()
 Timerplay.Enabled = False
End Sub

Private Sub timerplay_Timer()
'la proprietà Interval del timer deve essere impostata
                                          a priori
 numerofoto = numerofoto + 1
 If numerofoto <= listafile.Count Then
  framefoto.Picture = LoadPicture(pathdir & "\"
                         & listafile.Item(numerofoto))
  framefoto.Repaint
 'ridisegna il frame
 Else
  numerofoto = 0
 End If
End Sub
```

ZOOM E SPOSTAMENTO IMMAGINI

Per gestire lo zoom e lo spostamento delle immagini, sul Frame, bisogna innanzitutto impostare le proprietà che consentono al *Frame* di avere due *scrollBars (KeepScrollBarsVisible=3-fmScrollBarsBoth* e *ScroolBars=3-fmScrollBarsBoth)* e poi prevedere del codice simile al seguente.

```
Private Sub zoompiu_Click()
'Sub associata al pulsante +
 framefoto.Zoom = framefoto.Zoom + 10
 framefoto.Repaint
 …
End Sub
Private Sub zoomeno_Click()
'associata al pulsante -
 framefoto.Zoom = framefoto.Zoom - 10
 framefoto.Repaint
```

Oggetto Preview

Per caricare una immagine in un oggetto *Preview* (*Visualizzatore Immagini* di Windows XP) si può usare del codice simile al seguente:

```
MAX = App.path +
  "\foto\miafoto" + ".jpg"
Me.Preview1.Show (MAX)
```

Il visualizzatore è il *Preview1*, il metodo *Show* serve a mostrare l'immagine.

```
...
End Sub

Private Sub FrameFoto_Scroll(Private Sub framefoto_Scroll(
        ByVal ActionX As msforms.fmScrollAction, ByVal
    ActionY As msforms.fmScrollAction, ByVal RequestDx As
    Single, ByVal RequestDy As Single, ByVal ActualDx As
            msforms.ReturnSingle, ByVal ActualDy As
                        msforms.ReturnSingle)
 If ActionY = fmScrollActionLineDown Then
  ActualDy = -ActionY
  ActualDx = 0
 End If

 If ActionY = fmScrollActionLineUp Then
  ActualDy = ActionY
  ActualDx = 0
 End If

 If ActionX = fmScrollActionLineDown Then
  ActualDx = -ActionX
  ActualDy = 0
 End If

 If ActionX = fmScrollActionLineUp Then
  ActualDx = ActionX
  ActualDy = 0
 End If

End Sub
```

La procedura *FrameFoto_Scroll* serve per controllare i movimenti dell'immagine con gli scroollbars. Per capire come funzione il Frame della libreria *MsForms* vi consigliamo di leggere l'help in linea di Visual Basic.

LA DISEGNAFORM

Analizziamo vediamo la procedura che permette di disegnare il form *FrmTrasparente*:

```
Public Sub disegnaform()
 Dim CRegione As Long, Com As Long
 Dim WForm As Single, HForm As Single
 Dim WBordo As Single, HTitolo As Single
 Dim LControl As Single, TopControl As Single
 Dim WControl As Single, HControl As Single
 Dim Controllo As Control
 Dim Regione As Long
 Dim RegioneForm As Long
 WBordo = (Me.Width - Me.ScaleWidth) / 2
 HTitolo = Me.Height - Me.ScaleHeight - WBordo
 WBordo = ScaleX(WBordo, vbTwips, vbPixels)
 HTitolo = ScaleY(HTitolo, vbTwips, vbPixels)
 WForm = ScaleX(Me.Width, vbTwips, vbPixels)
 HForm = ScaleY(Me.Height, vbTwips, vbPixels)
 RegioneForm = CreateRectRgn(0, 0, WForm, HForm)
 Regione = CreateRectRgn(0, 0, 0, 0)
 Com = CombineRgn(Regione, RegioneForm,
                          RegioneForm, RGN_DIFF)
 For Each Controllo In Controls
  If (TypeOf Controllo Is CommandButton)
              Or Controllo.Name = "framefoto" Then
   LControl = ScaleX(Controllo.Left, vbTwips, vbPixels)
                                         + WBordo
   TopControl = ScaleX(Controllo.Top, vbTwips, vbPixels)
                                         + HTitolo
   WControl = ScaleX(Controllo.Width, vbTwips, vbPixels)
                                         + LControl
   HControl = ScaleX(Controllo.Height, vbTwips, vbPixels)
                                         + TopControl
   CRegione = CreateRectRgn(LControl, TopControl,
                                  WControl, HControl)
   Com = CombineRgn(Regione, Regione, CRegione,
                                           RGN_OR)
  End If
 Next Controllo
 SetWindowRgn hwnd, Regione, True
 Com = DeleteObject(Regione)
 Com = DeleteObject(CRegione)
 Com = DeleteObject(RegioneForm)
End Sub
```

Ricordiamo che i valori delle costanti sono

```
Public Const RGN_AND = 1
Public Const RGN_OR = 2
Public Const RGN_XOR = 3
Public Const RGN_DIFF = 4
...
```

La *DisegnaForm* (che deve essere richiamata nella *Form_Load*) è più efficiente della procedura descritta nel precedente articolo. Con *DisegnaForm* prima si calcola la superficie del Form, considerando anche le dimensioni del bordo e del titolo, poi si crea un'area che ricopre l'intero form e la si rende trasparente, infine si aggiungono le superfici dei controlli (dei *CommandButton* e del *FrameFoto*). Notate che le dimensioni sono convertite da *vbTwips* a *vbPixels* tramite la *ScaleX* e *ScaleY*.

CONCLUSIONI

L'applicazione presentata deve essere completata con le procedure che permettono di avviare la ricerca su internet e gestire il ridimensionamento dei form e dei controlli; inoltre bisognerebbe prevedere il codice che permette di catturare gli errori di *Run-Time*. Nel successivo appuntamento completeremo l'applicazione presentando le procedure che permettono di avviare la ricerca su internet e gestire il ridimensionamento dei form e dei controlli; inoltre presenteremo il codice per gestire gli errori di run-time.

Massimo Autiero

Visual Basic

Bibliografia

• MIGRARE DA MICROSOFT VISUAL BASIC 6.0 A MICROSOFT VISUAL BASIC .NET
Ed Robinson, Micheal Bond, Robert Ian Oliver

ISBN: 88-8331-349-6 (Mondadori Informatica)
Lingua: italiano
Pag.: 491 - € 45,00
2002
Contiene 1 CD-ROM
http://www.education.mondadori.it

Questo testo, come il titolo lascia intendere, si rivolge prevalentemente agli utilizzatori di Visual Basic 6. Gli autori introducono il lettore alle differenze tra la precedente versione del conosciuto ambiente di sviluppo visuale e la nuova versione per la piattaforma .NET. Dopo aver descritto le potenzialità della piattaforma .NET, il testo descrive dettagliatamente i passi da seguire per migrare le proprie applicazioni verso il nuovo ambiente. Sono descritti inoltre l'aggiornamento dei controlli ActiveX, componenti COM+ ed applicazioni distribuite. Il CD allegato contiene diversi esempi sia in Visual Basic 6.0 che in Visual Basic .NET ed in più la versione originale del testo in lingua inglese.

CINEMA4D

✓ Lo sviluppo di plug-in per tool grafici.

Maxon Cinema 4D

La programmazione di plugin per aggiungere nuove funzionalità è una delle prerogative più interessanti di Maxon Cinema. In questo articolo prenderemo in esame le varie problematiche realizzando un filtro per l'esportazione di modelli 3D.

Nello sviluppo di applicazioni basate su effetti grafici tridimensionali, in particolar modo dimostrativi e videogiochi, si sente spesso la necessità di importare modelli geometrici. Se le pretese non sono eccessive si può ricorrere ad un formato già noto (come il *3DS* di *3D Studio MAX)* scrivendo una routine per il caricamento delle varie informazioni (nel CD ne trovate una pronta all'uso). Sebbene sia la soluzione più facile molto spesso non è sufficiente, specialmente quando è necessario customizzare il tipo di elementi in gioco. Si impone, dunque, di scegliere se sviluppare un'applicazione in proprio o personalizzarne una già esistente inserendo le funzionalità di proprio interesse. La prima soluzione appare certamente più dispendiosa in termini di tempo e risorse, la seconda, invece, comporta una certa spesa iniziale ma anche un consistente risparmio di lavoro. Ogni pacchetto grafico mette infatti a disposizione un proprio kit per la programmazione di plugin, ovvero di un'applicazione esterna che inserisce nuove caratteristiche all'ambiente di sviluppo.

L'SDK DI CINEMA 4D

Cinema 4D offre un'ottima soluzione a questo genere di problemi proponendo un completo SDK *(Software Development Kit)* per lo sviluppo di plugin. Differentemente da altri package, l'intero SDK è facilmente reperibile tramite il Web, chiunque può scaricarlo dal sito *www.plugincafe.com* assieme ad altri validi strumenti di agevole installazione (help in linea, editor di risorse). Lo stesso sito offre un servizio di assistenza con un forum per le questioni tecniche ed un form per la registrazione del proprio plugin nel caso, una volta realizzato, si volesse distribuirlo. Cinema 4D, inoltre, garantisce un certo livello di astrazione dall'ambiente di sviluppo impiegato essendo disponibile su più piattaforme, è dunque l'ideale per chi ha in mente di intraprendere un progetto multi-piattaforma. A differenza di come si è portati a pensare, la programmazione di un plugin per Cinema 4D è un'operazione estremamente semplice, sono richiesti un livello medio di conoscenza del linguaggio C++ ed una preparazione basilare di computer grafica. Personalmente ho trovato il design e la struttura dell'SDK ben congegnato e di immediata fruibilità, certamente più schietto di altri offerti da software più blasonati. Per quanto riguarda l'ambiente di sviluppo, Maxon non impone particolari limitazioni ma offre supporto solo per Visual C++ e Metrowerks Codewarrior (su Macintosh), è nota anche la possibilità di utilizzare Borland C (la documentazione a proposito però è in lingua tedesca). Per la cronaca, esiste un linguaggio interno simile al C++ e notevolmente semplificato: il *COFFEE*. Tuttavia, sebbene sia documentato, non c'è un valido IDE per cui è preferibile seguire la strada del C++.

PREPARIAMO L'AMBIENTE DI SVILUPPO

Per prima cosa si consiglia di installare l'SDK seguendo fedelmente le istruzioni allegate facendo a priori alcune prove per familiarizzare con l'ambiente. Nella realizzazione di un plugin, anziché creare un nuovo progetto partendo da zero, può risultare comodo copiare la cartella dell'SDK stesso e lavorare all'interno della stessa. L'SDK si presenta con una serie di plugin di base per gli usi più comuni, in diversi casi può essere sufficiente eliminare le parti superflue cancellando a livello di codice le relative voci ed inclusioni. Una volta effettuata una copia della cartella CINEMA4DSDK dovrete eliminare tutte quelle contenute tranne *"api"*, *"filter"* (eliminando solo il contenuto), *"obj" (se c'è)*, *"res"* e *"string_us"*. Queste ultime due contengono informazioni su finestre di dialog e tavole di stringhe, tuttavia se dovete farne un uso semplice (come nel nostro caso) vi consiglio di non crearne nuove e modificare manualmente i file già presenti con un normale editor di testi, altrimenti potete scaricare un editor di risorse da plugincafé. Per fare funzionare il plugin presente nel CD-

Fig. 1: **Assicurarsi di lasciare integri almeno questi file.**

ROM è necessario effettuare da Windows un'operazione di copia-incolla verso questa cartella, sostituendo i file ove richiesto; successivamente sarà possibile aprire il progetto e costruire il plugin.

STRUTTURA E REALIZZAZIONE DEL PLUGIN

Andrebbe ben oltre i nostri scopi trattare in questa sede i diversi tipi di plugin supportati, pertanto restringeremo il nostro campo di interesse alla creazione di un filtro per esportare gli oggetti presenti nello scenario, salvando le informazioni secondo un nostro formato testuale. Ci serviremo dunque dei sorgenti inclusi nel CD allegato al presente numero che consigliamo di installare con attenzione. Ogni plugin deve essere costruito sulla struttura di una delle classi messe a disposizione dall'SDK e specifica per il tipo di applicazione da realizzare. Per il nostro caso impieghiamo *FILTER3DPLUGIN* della quale ci interessano i due membri coinvolti nell'esportazione: *Suffix* e *Save*. Il primo è un puntatore a stringa contenente il suffisso del nome file, il secondo è il puntatore alla funzione che si occuperà dell'intero procedimento di esportazione. E' giusto far notare che, al fine di conservare la compatibilità con le varie piattaforme, il pacchetto SDK mette a disposizione una serie di tipi di dati proprietari, la stringa con la quale ci troviamo ad operare non è la stessa di MFC o della libreria standard di C++, segue delle proprie regole. Studiando il file *Main.cpp* saltano all'occhio due funzioni:

```
LONG C4D_PlStart(void *p1, void *p2){
   if (!resource.Init()) return FALSE;
   // filter plugins
   if (!RegisterExporter()) return FALSE;
   return TRUE;
}
void C4D_PlEnd(void)
{ FreeExporter();}
```

Esse possono essere viste rispettivamente come il costruttore e il distruttore di una classe. *C4D_PlStart()*, simile alla *main()* del linguaggio C, serve ad allocare le risorse ed inizializzare il plugin, *C4D_PlEnd()* è deputata al rilascio della memoria utilizzata. *RegisterExporter()* ha lo scopo di registrare il plugin nell'ambito di Cinema 4D:

```
static String *str=NULL;
Bool RegisterExporter(void) {
String name=GeLoadString(IDS_EXPORTER); if
                    (!name.Content()) return TRUE;
FILTER3DPLUGIN cs;
ClearMem(&cs,sizeof(cs));
str = gNew String("iop"); if (!str) return FALSE;
cs.Save = ExporterSave;
cs.suffix = str->Get();
return GeRegisterPlugin(C4DPL_FILTER3D, ID, name,
                              &cs, sizeof(cs));}
```

Come premesso, nel membro *Save* di *cs* viene scritto l'indirizzo della funzione di esportazione:

```
static LONG ExporterSave(_Filename *oname,
                         _BaseContainer *settings,
BaseDocument *doc, LONG flags) {
if (!(flags&FILTER3D_OBJECTS)) return FILEERROR_NONE;
Filename name = Filename(oname,FALSE);
C3DModel model(settings, doc, name);
model.DoExport();
if(model.SaveFile() == FALSE)
return FILEERROR_MEMORY;
return FILEERROR_NONE; }
```

Per motivi di convenienza l'intero programma è stato racchiuso all'interno di una classe, *C3DModel*, che in realtà è solo una derivazione di un'altra: *CExporter*. Quest'ultima contiene il codice di tutte le operazioni più importanti del plugin vero e proprio, come la visita dei nodi, la lettura degli attributi e la catalogazione dei materiali; possiamo definire *CExporter* il cuore dell'applicazione (Fig. 2). In realtà, non avrebbe senso utilizzare direttamente *CExporter* per scrivere un plugin, essa infatti non produce alcun output, si limita ad individuare, all'interno di Cinema 4D, gli oggetti interessanti "segnalandoli" tramite una serie di funzioni virtuali. Per questo motivo, se si intende scrivere un'applicazione funzionante, è necessario creare una nuova classe che la erediti e specializzare le parti che servono. Il funzionamento è tutt'altro che complicato: quando la procedura si imbatte in un oggetto, esso non viene direttamente letto e scritto su file ma esaminato per passare i suoi dati fondamentali ad una funzione virtuale che,

Fig. 2: **C3DModel deriva da CExporter, il cuore dell'applicazione.**

CINEMA4D

Plugin sul CD-Rom

Per fare funzionare il plugin presente nel CD-ROM è necessario effettuare da Windows un'operazione di copia-incolla verso questa cartella sostituendo i file ove richiesto; successivamente sarà possibile aprire il progetto e costruire il plugin.

CINEMA4D

una volta implementata, si occuperà di trattarli per come necessario. Ad operazione compiuta, dove serve, ne viene chiamata un'altra per informare che l'elaborazione ha avuto termine. Nell'intervallo tra l'esecuzione di questi due tipi di funzioni vengono richiamate quelle che passano al programma i contenuti veri e propri dell'elemento in esame (nel caso di un oggetto: i vertici, le facce, il materiale assegnato, etc.). Nella maggior parte dei casi, per realizzare un vostro plugin di esportazione, non dovrete modificare il codice ma semplicemente derivare una nuova classe da *CExporter* premurandovi di implementare le seguenti funzioni:

1) *OnObjectFound()*
2) *OnObjectGeometry()*
3) *OnObjectMaterial()*
4) *OnUVWTag()*
5) *OnObjectEnd()*
6) *OnMaterialFound()*
7) *OnReadingColorChannel()*
8) *OnChanData()*
9) *OnMaterialEnd()*

Per le dichiarazioni complete delle funzioni si faccia riferimento ai sorgenti allegati; in questo articolo daremo una spiegazione del loro significato. *OnObjectFound()* e *OnObjectEnd()* sono chiamate rispettivamente quando un oggetto poligonale viene individuato e quando la sua interpretazione ha avuto termine. Tra gli argomenti vengono passati il nome, matrice di trasformazione e livello nella gerarchia, secondo l'ordine di catalogazione. Le funzioni comprese tra *OnObjectFound()* e *OnObjectEnd()* forniscono le informazioni sulla struttura geometrica, il particolar modo *OnObjectGeometry()* passa i due array contenenti vertici e poligoni, *OnObjectMaterial()* il materiale associato ed infine *OnUVWTag()* una particolare struttura che permette di individuare per ogni vertice dei poligoni le corrispondenti coordinate *U,V,W* della texture. Un discorso molto simile vale per *OnMaterialFound()* e *OnMaterialEnd()*, la prima però, oltre a segnalare il nome del materiale, fornisce anche un *Marker:* in Cinema 4D è un identificatore univoco, può mostrarsi utile in svariate circostanze dato che l'uso dei soli nomi per orientarsi potrebbe comportare ambiguità. Ogni materiale è dotato di un numero abbastanza ampio informazioni su colori e texture, queste sono distribuite su diversi canali, ognuno dei quali distinto per ciò che rappresenta. In questo nostro esempio ci limitiamo ad esaminare il canale *COLOR.* A questo proposito, *OnChanData()* passa una struttura dati definita all'interno del plugin, contenente i vari parametri:

```
typedef struct _ChannelInfo {
Vector Color;
Real Brightness;
String Texture;
Real BlurOffset, BlurStrength;
LONG Interpolation, MixMode;
Real MixStrength;
LONG TimeFrom, TimeTo, TimeFPS, TimeMode, TimeTiming;
LONG TimeLoops, ShaderID;}
ChannelInfo;
```

Materiali

Ogni materiale è dotato di un numero abbastanza ampio informazioni su colori e texture, queste sono distribuite su diversi canali ognuno dei quali distinto per ciò che rappresenta.

ChannelInfo riporta i dati di maggiore interesse quali il colore, rappresentato qui tramite il vettore *Color*, e il nome del file di texture. Le funzioni elencate vengono richiamate in automatico ogni volta che un particolare elemento è stato riscontrato, ragion per cui la programmazione di un plugin per l'esportazione si riduce alla corretta catalogazione delle informazioni ottenute di volta in volta e al salvataggio su disco. La nostra versione di *C3DModel* si comporta esattamente in questa maniera, come potete vedere la quasi totalità del codice è impiegato nella gestione dei dati ricavati e la loro serializzazione su disco tramite il file stream di C++. In questo paragrafo abbiamo fatto una panoramica delle classi coinvolte nel procedimento di esportazione studiandone l'impiego. Ora che sappiamo come utilizzarle cercheremo di andare più a fondo per comprendere il funzionamento

I MATERIALI

Il procedimento di analisi viene avviato con il metodo *DoExport()* di CExporter che richiama, nel giusto ordine, le funzioni delegate ad estrarre le informazioni.
Come prima cosa vengono passati in rassegna i materiali coinvolti nella scena, di questo se ne occupano *DoHeader()*, *BrowseMaterials()* e *ReadMaterial()*, vediamo come. Si faccia caso alla dichiarazione della funzione di salvataggio di *FILTER3DPLUGIN:*

```
LONG (*Save )(_Filename* name, _BaseContainer*
        settings, BaseDocument* doc, LONG flags)
```

Tra gli argomenti notiamo un puntatore ad un oggetto di tipo *BaseDocument*. Questo mette a disposizione gli strumenti necessari per ricavare informazioni sui dati contenuti nella scena. Per ricavare il primo dei materiali, infatti, è sufficiente ricorrere al metodo *GetFirstMaterial():*

```
// Preleviamo il primo materiale assegnato al documento
BaseMaterial *baseMaterialPtr = doc->GetFirstMaterial();
if (baseMaterialPtr == NULL)
return FALSE;
```

Gli altri, se ve ne sono, possono essere ottenuti adoperando il metodo *next()* di *BaseMaterial:*

```
BaseMaterial* next(void)
```

Per facilitare l'operazione possiamo scorrere la lista con una funzione ricorsiva:

```
Bool BrowseMaterials(BaseMaterial *baseMaterialPtr){
// Andiamo avanti nella lista dei materiali
baseMaterialPtr = baseMaterialPtr->next();
if(baseMaterialPtr != NULL)
BrowseMaterials(baseMaterialPtr);
// Se ve ne è un altro, procedi
return TRUE;}
```

CINEMA4D

Una volta individuato il materiale possiamo ricavarne facilmente il nome.

```
// Preleviamo il nome
String materialName;
materialName = baseMaterialPtr->GetName();
```

Fortunatamente *BaseMaterial* deriva da *BaseList2D* e possiamo ricavare un identificatore, il *Marker*. Come accennato, il *Marker* è molto utile in quanto più materiali potrebbero avere lo stesso nome ed allora sarebbe necessario servirsi dell'identificatore univoco.

```
Marker materialMarker;
materialMarker = baseMaterialPtr->GetMarker();
```

Un compito leggermente più complicato è destinato alla lettura delle informazioni contenute, dato che queste sono suddivise in più canali. Per il nostro caso ci interessa solo il canale *COLOR*.

```
// Occupiamoci dei canali
BaseChannel *baseChannelPtr;
baseChannelPtr = baseMaterialPtr->GetChannel(
                                    CHANNEL_COLOR);
```

Il metodo *GetChannel()* fornisce un puntatore all'oggetto *BaseChannel* contenente i dati di nostro interesse, è comunque opportuno verificare l'effettiva esistenza del canale cercato controllando che l'indirizzo non sia nullo. Le informazioni che stiamo cercando non sono direttamente disponibili nell'oggetto che abbiamo ottenuto, esse - come spesso avviene in Cinema 4D - sono memorizzate in un container, definito dalla classe *BaseContainer*. Un container, traducendo la definizione fornita dal SDK, è una collezione di valori individuali ciascuno dei quali dotato di un proprio tipo e identificatore; per accedere alle informazioni contenute nel container bisogna utilizzare delle funzioni specifiche per il tipo di dato che si intende trattare. Per accedere al container useremo il metodo *GetData()*, quindi *GetVector()* e *GetString()* per ottenere il colore ed il nome della texture. Di seguito viene proposto un esempio:

```
ChannelInfo chanInfo;
ClearMem(&chanInfo, sizeof(chanInfo));
// Leggiamo ora i dati del canale
chanInfo.Color = baseChannelPtr->
          GetData().GetVector(BASECHANNEL_COLOR);
chanInfo.Texture = baseChannelPtr->
        GetData().GetString(BASECHANNEL_TEXTURE);
```

GLI OGGETTI

Gli oggetti di cui è composta la scena possono essere trattati in un modo molto simile, sempre tramite BaseDocument possiamo accedere al primo con il metodo *GetFirstObject()* ed impiegare *next()* per passare al successivo, tuttavia il discorso va esteso perché gli oggetti possono seguire una disposizione gerarchica e ogni nodo può contenere dei figli; in questo ci viene in aiuto la funzione *down()* che ricava il primo figlio associato:

```
void BrowsePolyObjects(BaseObject *ObjectPtr, Matrix
                          levelUpMatrix, INT deepLevel)
{
  Matrix objMatrix;
  do
  {
    objMatrix = levelUpMatrix * ObjectPtr->GetMl();
    BaseObject *objPtr = ObjectPtr->down();
    if(objPtr != NULL)
      BrowsePolyObjects(objPtr, objMatrix, deepLevel + 1);
      ObjectPtr = ObjectPtr->next();
  }
  while(ObjectPtr != NULL);
}
```

La routine esamina i vari oggetti uno per uno ricavando la matrice di trasformazione che, di volta in volta, viene passata con l'argomento *levelUpMatrix*; deepLevel indica il livello di profondità nella gerarchia. Una volta ottenuti i vari oggetti che possono utilizzare *GetPoint()* e *GetPolygon()* per ricavare le informazioni su vertici e poligoni, il grosso è fatto; per il resto si può fare riferimento ai sorgenti allegati.

CONCLUSIONI

Siamo giunti al termine di questo appuntamento, spero che le informazioni contenute nell'articolo ed i sorgenti allegati siano utili nell'approfondimento di quest'argomento che sono certo non mancherà di darvi immediate soddisfazioni; personalmente ho trovato la programmazione di plugin per Cinema 4D estremamente divertente ed utile. Nel CD, oltre al codice del plugin, è presente un lettore per il formato dei file esportati (Fig.3) con visualizzazione tramite OpenGL. Buon lavoro!

Andrea Ingegneri

Fig. 3: Disegno con OpenGL di un modello esportato.

GetChannel()

Il metodo *GetChannel()* fornisce un puntatore all'oggetto *BaseChannel* contenente i dati di nostro interesse, è comunque opportuno verificare l'effettiva esistenza del canale cercato controllando che l'indirizzo non sia nullo.

Pocket PC

La comunicazione con un application server di un sistema mobile

GPS e PocketPC: la logica server

Un Application Server nelle soluzioni che includono client PockePC rappresenta un attore fondamentale nella logica funzionale dell'intero processo informatico.

Requisiti

HARDWARE: Pocket PC o (dispositivo Windows CE Based), Computer con almeno processore Pentium II e 128 MB di memoria, un modulo GPS per Pocket PC.

SOFTWARE: Windows 98 SE /2000/XP, IIS (oppure PWS-Personal WEB Server con Windows 9x), SQL Server 2000 con Service Pack 1 o superiore, SQL Server CE 2.0, Embedded Visual Tools.

Nel precedente articolo abbiamo progettato una applicazione che si interfaccia con un modulo GPS esterno al Pocket PC. Sono stati visti i passi fondamentali tramite i quali è possibile comunicare con il dispositivo esterno al Pocket PC e leggere i dati provenienti dal satellite. Nel presente articolo ci proponiamo di progettare e implementare la parte Server dell'applicazione che ha lo scopo di visualizzare la posizione di un veicolo equipaggiato con il Pocket PC e il relativo modulo GPS. Anche la parte Client dovrà essere modificata in modo tale da aggiungere la logica necessaria per l'accesso ai dati e per la sincronizzazione. In particolare, una volta che avremo progettato il Database dell'applicazione sarà necessario effettuare la pubblicazione dello stesso con la procedura illustrata in un precedente articolo di ioProgrammo. La pubblicazione permetterà alla logica di sottoscrizione e sincronizzazione sul palmare di creare una replica del database sul client e di sincronizzarne i dati. La logica di accesso ai dati si occuperà non solo di recuperare i dati provenienti dal server ma anche di inserirvi i dati di posizione che rileviamo dal satellite con il modulo GPS. Lasciamo al lettore il compito di creare queste due classi facendo riferimento alle tecniche viste in precedenza.

PROGETTAZIONE DELLO SCHEMA DEL DATABASE

In Fig. 1. riportiamo lo schema del database che chiameremo *PositionLocator* e che costruiremo su SQL Server 2000. Nella stessa figura è anche possibile reperire

TAB_POSIZIONI

Column Name	Data Type	Length	Allow Nulls
ID_POSIZIONE	uniqueidentifie	16	
ID_VEICOLO_FK	uniqueidentifie	16	✓
LATITUDINE	float	8	
LONGITUDINE	float	8	
DATA	datetime	8	
DESCRIZIONE	nvarchar	250	
SYNCH_FLAG	bit	1	✓

TAB_VEICOLI

Column Name	Data Type	Length	Allow Nulls
ID_VEICOLO	uniqueidentifie	16	
DESCRIZIONE	nvarchar	100	
STATO	nvarchar	50	✓
ID_TIPOLOGIA_VEICC	uniqueidentifie	16	✓
USERID	nvarchar	10	✓
PSW	nvarchar	10	✓

TAB_TIPOLOGIE_VEICOLO

Column Name	Data Type	Length	Allow Nulls
ID_TIPOLOGIA_VEICC	uniqueidentifie	16	
DESCRIZIONE	nvarchar	50	

Fig. 1: Schema del database.

le informazioni corrispondenti ai tipi dei campi. Lo schema comprende tre tabelle, corrispondenti alla minima quantità di informazioni da gestire nel nostro problema di posizionamento. E' chiaro che lo schema potrà essere arricchito a seconda delle funzionalità che dovrà rendere disponibile la nostra applicazione e della tipologia di dati da scambiare tra i vari attori. La tabella *TAB_VEICOLI*, mantiene le informazioni relative al veicolo di cui vogliamo controllare la posizione. In particolare, abbiamo modellato il problema in maniera tale che ad ogni veicolo è associato un account. Quindi, se disponiamo di un insieme di veicoli con il campo *DESCRIZIONE* uguale a "auto1", "auto2", etc… affinchè si possano rilevare le posizioni del veicolo "auto1" dobbiamo inserire nella fase di login dell'applicazione palmare le corrette userid e password di quel veicolo, facendo attenzione a non inserire quelli relative a quelle del veicolo "auto2" altrimenti il sistema associerà le coordinate al veicolo sbagliato. Il campo *ID_VEICOLO* è di tipo uniqueidentifier e rappresenta la chiave principale per la tabella. La tabella *TAB_TIPOLOGIE_VEICOLO,* ha due soli campi: una chiave di tipo *uniqueidentifier* e una descrizione testuale. Questa tabella ha lo scopo di memorizzare la tipologia di veicolo di cui vogliamo controllare la posizione. Ad esempio, per applicazioni in ambito ospedaliero potremmo voler suddividere le categorie di veicoli, in ambulanze per il soccorso ordinario e in elicotteri per il soccorso straordinario. La tabella *TAB_VEICOLI* ha un vincolo di chiave esterna con la tabella in questione con il campo *ID_TIPOLOGIA_VEICOLO_FK.*

Nella tabella *TAB_POSIZIONI*, verranno memorizzate le coordinate del veicolo in movimento. In particolare, i campi *LATITUDINE* e *LONGITUDINE* rappresentano le coordinate rilevate nell'istante memorizzato nel campo *DATA* dello stesso record. Inoltre, possiamo notare il campo *ID_VEICOLO_FK* che rappresenta una chiave esterna per la tabella *TAB_VEICOLI* e permette di associare le coordinate correnti al veicolo corrispondente.

Pocket PC

PARTE SERVER DELL'APPLICAZIONE

Svilupperemo la parte Server dell'applicazione fornendo due soluzioni distinte:

1. prima presenteremo una applicazione in tecnologia .NET sviluppata come Windows Application che utilizza il componente ActiveX installato insieme al prodotto Microsoft MapPoint 2002.
2. Illustreremo quindi una semplice WEB Application in tecnologia .NET e che utilizza il Web Service di MapPoint.NET di Microsoft per il quale disponiamo di un account a tempo limitato ma sufficiente per testare la nostra applicazione.

Svilupperemo la logica necessaria per consentire ad un utente di visualizzare la posizione di un veicolo che abbia sincronizzato la sua posizione al Server di Database. Inoltre, ci proponiamo di implementare la funzionalità di ricerca del veicolo che sia posizionato ad una distanza minima rispetto ad un dato luogo di cui conosciamo le coordinate o solo una sua descrizione. Microsoft MapPoint 2002 si configura come applicazione con molteplici funzionalità. Questo prodotto Microsoft rende disponibile un componente ActiveX, utilizzabile solo all'interno di una Windows application, che consente di interfacciarsi con l'*Object Model* di *MapPoint 2002* e di visualizzare le mappe di interesse.

SVILUPPO DELLA WINDOWS APPLICATION

Nel CD di "io Programmo" rilasciamo l'ultima versione della Windows Application *PositionLocatorServer*. In Fig. 3 possiamo notare la schermata iniziale della applicazione. Possiamo osservare alcuni oggetti interessanti nella maschera:

1. Un controllo *TreeView* che verrà caricato con i veicoli di cui vogliamo controllare le posizioni sulla mappa;
2. Il controllo ActiveX con il quale riusciamo a visualizzare la mappe e ad interfacciarci con lo *Object Model di MapPoint 2002*.
3. Il Tasto "Cerca località" che permetterà di visualizzare un'altra maschera con la quale potremo ricercare un luogo di interesse inserendo una posizione e visulizzare il veicolo più vicino al luogo trovato con le relative indicazioni stradali.

Fig. 2: **Windows Application di posizionamento.**

La Windows Form nel workspace corrispondente alla maschera di Fig. 2 è *frmListaVeicoli.vb*, vediamo il costruttore della relativa classe:

```
Me.mapCtrl.NewMap(MapPoint.GeoMapRegion.
                                        geoMapEurope)
    objDataAccess = New DataAccessLayer()
    FillTreView()
```

Nel codice si evince come venga inizializzato il controllo ActiveX di MapPoint 2002 alla mappa dell'Europa; inoltre viene creata l'istanza dell'oggetto *objDataAccess* della classe *DataAccessLayer*. Rimadiamo il lettore ad una attenta analisi di quest'ultima classe. La classe *DataAccessLayer* fornisce tutte le funzionalità necessarie per l'accesso al Database SQL Server e il recupero dei dati di posizione e dei veicoli. Infine, viene richiamata la procedura *FillTreeView* con la quale viene popolato il *TreeView* con le descrizioni dei veicoli. L'implementazione di questa ultima procedura è molto semplice:

```
objDataAccess.GiveVeicoliInfo(TreeView1)
```

Fig. 3: **Visualizzazione di un punto sulla mappa.**

Possiamo notare che viene richiamato un metodo della classe *DataAccessLayer* a cui si passa il riferimento alla variabile membro *TreeView1*. Per la sua implementazione rimandiamo al codice presente nel CD proponendoci di concentrare la nostra attenzione sulla logica relativa al rendering delle posizioni sulla mappa. Possiamo notare dalla Fig. 2 che i veicoli sono stati suddivisi per categoria; quindi abbiamo 2 veicoli che appartengono, ad esempio, alla categoria ambulanze e un mezzo mobile appartenente alla categoria elicottero. Questa suddivisione dei mezzi mobili ci ha consentito di costruire la funzionalità per mezzo della quale se effettuo un click sulla categoria di veicoli potrò visualizzare sulla mappa l'ultima posizione sincronizzata da tutti i mezzi mobili appartenenti a quella categoria come la figura stessa dimostra. Allo stesso modo il click sulla descrizione di un veicolo permetterà di visualizzare sulla mappa l'ultima posizione del veicolo stesso (Fig. 3). Vediamo il codice necessario per implementa-

Lato server

La parte Server dell'applicazione è stata sviluppata con tecnologia .NET secondo due soluzioni: una Windows Application che utilizza l'ActiveX di MapPoint 2002 e una WEB Application che utilizza il WEB Service MapPoint .NET.
La prima soluzione, pur nella sua complessità, potrebbe risultare meno costosa della seconda. Infatti il costo del prodotto MapPoint 2002 potrebbe essere non paragonabile a quello di acquisto del WEB Service MapPoint.NET che dipende dal traffico di dati. Bisogna precisare che il vantaggio della seconda soluzione è il fatto che, essendo una WEB Application, una volta pubblicata sul WEB potrebbe essere accessibile da ogni luogo che presenti una connessione alla rete. E' chiaro che bisogna fare le dovute considerazioni in base al numero di utilizzatori e/o dal luogo in cui tali informazioni devono essere reperibili. Un altro fattore non trascurabile è il flusso di dati nell'unità di tempo.

Pocket PC

Rendering

La logica di rendering della procedura *VisulizzaPuntoDaCoordinate* (visualizzazione di un solo punto sulla mappa) presenta una differenza sostanziale rispetto a *VisualizzaVeicoliPerCategoria* (visualizzazione di più punti). Nella classe di Accesso ai Dati (classe DataAccessLayer nel Workspace), nella fase di popolazione del treeview dei veicoli, è stata popolata una lista contenente le informazioni dei veicoli che sono stati censiti nell'applicazione. Per questa lista sono disponibili metodi che permettono di recuperare i dati di posizione relativi ad ognuno degli suoi elementi; ad esempio il metodo *GetItemForIndex (i, objInfoVeicolo)* pemette di valorizzare in maniera opportuna le property dell'istanza *objInfoVeicolo* della classe *InfoVeicolo* relativo all'*i-esimo* elemento. Dal codice della procedura precedente si vede come questo metodo sia richiamato nel ciclo while allo scopo di ricavare l'ultima posizione sincronizzata per ogni veicolo e visualizzata sulla stessa istanza della classe Map. L'accortezza di utilizzare la stessa istanza della mappa consente di sovrapporre la visualizzazione di più oggetti di tipo Pushpin. Il risultato della procedura può essere visualizzato selezionando a runtime una qualunque categoria di veicoli dal treeview come viene mostrato in Fig. 2.

re queste funzionalità. E' appena il caso di evidenziare che bisogna gestire l'evento di selezione di un nodo di un controllo *TreieView*, ecco l'implementazione del gestore dell'evento *AfterSelect* del treeview:

```
Dim myNode As VeicoloNode = e.Node
Dim objInfoVeicolo As InfoVeicolo = New InfoVeicolo()
Dim latitOld As Double = myNode.Latitudine
Dim longOld As Double = myNode.Longitudine
Dim latitNew As Double
Dim longNew As Double
Dim descrNew As String
If Not (myNode.IsCategoria) Then
    objDataAccess.GiveMeInstantPosition(
                    myNode.Guid_Veicolo, objInfoVeicolo)
    atitNew = objInfoVeicolo.Latitudine
    longNew = objInfoVeicolo.Longitudine
If (Not (latitOld = latitNew) And Not (longOld =
                                        longNew)) Then
  myNode.Latitudine = latitNew
  myNode.Longitudine = longNew
  myNode.Descrizione_Posizione = descrNew
  VisualizzaPuntoDaCoordinate(myNode.Longitudine,
      myNode.Latitudine, myNode.Descrizione_Veicolo, _
                         myNode.Descrizione_Posizione)
Else
  VisualizzaPuntoDaCoordinate(myNode.Longitudine,
        myNode.Latitudine, myNode.Descrizione_Veicolo,
                         myNode.Descrizione_Posizione)
End If
ElseIf (myNode.IsExpanded) Then
  VisualizzaVeicoliPerCategoria(myNode.Tipo_Veicolo)
End If
```

Possiamo notare dal codice che dal secondo argomento del gestore di evento si riesce a recuperare l'informazione del nodo selezionato. Nel Workspace dell'applicazione possiamo notare la classe *VeicoloNode* (che estende *TreeNode*) che rappresenta la classe base dei nodi che compongono un controllo *TreeView*. Dopo queste considerazioni la descrizione della logica di rendering appare più semplice: prima di tutto vengono recuperate le informazioni del nodo selezionato e le memorizziamo nella variabile *myNode*. A questo punto viene fatto un controllo se il nodo in questione si riferisce ad un veicolo oppure ad una categoria di veicoli. Nel primo caso, utilizziamo la procedura *GiveMeInstantPosition* della classe *DataAccessLayer* la quale riceve in ingresso il *GUID* che identifica univocamente un veicolo nel database e un oggetto di tipo *InfoVeicolo* in cui vengono memorizzate le informazioni sul veicolo. Lo scopo di questa funzione è quella di rilevare le ultime coordinate sincronizzate dal client Pocket PC per il *Veicolo* e quindi valorizzare l'istanza *objInfoVeicolo* della classe *VeicoloInfo*. Una volta recuperate le ultime coordinate viene visualizzata sulla mappa la posizione del veicolo con la procedura *VisulizzaPuntoDaCoordinate* (…). Questa funzione riceve i dati di latitudine e longitudine del veicolo, oltre ad una descrizione del veicolo stesso e del luogo da visualizzare. Nel caso in cui il nodo corrente selezionato sia una categoria vengono visualizzati sulla mappa tutti i veicoli appartenenti a quella categoria.

RENDERING SULLA MAPPA

A questo punto vediamo il dettaglio delle procedure della classe *frmListaVeicoli* che consentono la visualizzazione sulla mappa del componente ActiveX di Map-Point della posizione di un singolo veicolo o di un gruppo di veicoli appartenenti ad una stessa categoria. Cominciamo con l'analizzare la prima tipologia di rendering. Di seguito riportiamo il codice della procedura *VisulizzaPuntoDaCoordinate(…):*

```
Dim oMap As MapPoint.Map
Dim oPush As MapPoint.Pushpin
Dim oLoc As MapPoint.Location
oMap = Me.mapCtrl.ActiveMap
oMap.Saved = True
Me.mapCtrl.NewMap(MapPoint.GeoMapRegion.geoMapEurope)
oMap = Me.mapCtrl.ActiveMap
oLoc = oMap.GetLocation(Latit, Longit)
oPush = oMap.AddPushpin(oLoc, …)
oPush.BalloonState =
             MapPoint.GeoBalloonState.geoDisplayBalloon
oPush.Highlight = True
oMap.DataSets.ZoomTo()
oMap.Saved = True
End Sub
```

Dalle prime linee di codice possiamo prendere visione di alcuni importanti oggetti dell'Object Model di Map-Point 2002. Prima di tutto dichiariamo un oggetto *oMap* di tipo *Map* che rappresenta proprio la mappa su cui andremo a disegnare tramite le procedure del componente. L'oggetto *oMap* verrà inizializzato con l'istanza ritornata dal metodo *GetActiveMap()* invocato sulla istanza componente ActiveX *mapCtrl*. Gli altri due oggetti dichiarati sono i seguenti:

- **Loc** : rappresenta un oggetto della classe *Location* in cui vengono memorizzati le coordinate di un veicolo; esso viene, infatti, inizializzato con il risultato della invocazione della funzione *GetLocation* sull'istanza dell'oggetto *oMap* passandovi le coordinate come parametri di ingresso.
- **Push:** è un oggetto della classe *Pushpin*. Una istanza di questa classe viene ottenuto invocando la funzione *AddPushpin* sulla istanza *oMap* e passando come parametro di ingresso l'oggetto *oLoc* creato prima e come secondo parametro una descrizione da associare alla posizione sulla mappa. In pratica, un *Pushpin* rappresenta una icona da associare ad una specifica posizione. L'invocazione dei metodi *oPush.Highlight = True* e *oMap.DataSets.ZoomTo()* permetteranno di rendere visibile l'icona del punto sulla mappa e di centrarla sul punto stesso per una

migliore visualizzazione. L'ultima istruzione

- **Map.Saved = True**, consente di eliminare la maschera che ci chiede di salvare i cambiamenti sulla mappa corrente.

Compresa la logica di rendering di una sola posizione, è molto semplice comprendere quella relativa al rendering di differenti posizioni sulla stessa mappa. Di seguito riportiamo il codice della procedura *VisualizzaVeicoliPerCategoria* necessario alla visualizzazione di più veicoli:

```
...
oMap = Me.mapCtrl.ActiveMap
    oMap.Saved = True
    Me.mapCtrl.NewMap(MapPoint.GeoMapRegion.
                                    geoMapEurope)
    oMap = Me.mapCtrl.ActiveMap
    Dim i As Integer = 0
    Dim objInfoVeicolo As InfoVeicolo = New InfoVeicolo()
  Do While (i <= objDataAccess.GetVeicoliCount() - 1)
      objDataAccess.GetItemForIndex(i, objInfoVeicolo)
      If Not (objInfoVeicolo Is Nothing) Then
        If objInfoVeicolo.Tipo_Veicolo = tipoVeicolo Then
oLoc = oMap.GetLocation(objInfoVeicolo.Latitudine,
                             objInfoVeicolo.Longitudine)
    oPush = oMap.AddPushpin(oLoc, ..)
    oPush.BalloonState = MapPoint.GeoBalloonState.
                                       geoDisplayBalloon
    oPush.Highlight = True
    oMap.DataSets.ZoomTo()
    oMap.Saved = True
End If
End If
i = i + 1
Loop
```

La logica di rendering presenta una differenza sostanziale rispetto alla precedente.

FUNZIONALITÀ DI RICERCA DI UN LUOGO

Di seguito riportiamo parte del codice della funzione di ricerca di un luogo sulla mappa:

```
...
    Dim oPush As MapPoint.Pushpin
    Dim latitudine As Double, longitudine As Double
    Dim Descrizione As String
    Dim distMinima As Double
    GetLatLong(Me.txtStreet.Text, Me.txtCity.Text,
         Me.txtState.Text, latitudine, longitudine, oPush)
    Dim objInfoVeicolo As InfoVeicolo
    oMap = Me.MapCtrl.ActiveMap
    Me.m_locArrivo = oMap.GetLocation(latitudine,
                                       longitudine, 100)
    objDataAccess.GetVehicleMostNear(latitudine,
           longitudine, distMinima, objInfoVeicolo, oMap)
    Me.m_locPartenza = oMap.GetLocation(
                        objInfoVeicolo.Latitudine, _
                   objInfoVeicolo.Longitudine, 100)
    VisulizzaMinimo(longitudine, latitudine,
                               objInfoVeicolo, oPush)
  End Sub
```

La logica è piuttosto semplice: viene prima controllato che l'utente abbia inserito le informazioni sul luogo da ricercare. Il passo successivo sarà quello di utilizzare la funzione *GeLatLong()* nella quale è implementata la ricerca vera e propria utilizzato i metodi dell'Object Model di MapPoint. Una volta che abbiamo ricavato le coordinate del luogo ricercato utilizziamo queste informazioni per istanziare un oggetto di tipo Location per la successiva fase di rendering sulla mappa. A questo punto viene ricavato il veicolo più vicino alla posizione ricercata utilizzando l'oggetto *objDataAccess* che espone il metodo *GetVehicleMostNear(..)*; questo metodo ci fornisce in uscita le informazioni sul veicolo nella variabile *objInfoVeicolo* e la distanza a cui si trova nella variabile *distMinima*. Alla fine visualizziamo le due posizioni sulla stessa mappa. È interessante, a questo punto, illustrare la logica della funzione *GetVehicleMostNear* della classe di accesso ai dati, rimandando al codice nel CD per la sua implementazione: viene iterata la lista dei veicoli e viene trovato quello per cui la sua ultima posizione sincronizzata sia a "distanza minima" dalla posizione di interesse. La funzionalità di distanza viene implementata in MapPoint con il metodo *Distance* dell'Oggetto Map; questo metodo riceve due oggetti di tipo *Location* e restituisce la distanza tra di essi in base ai dati relativi alla mappa.

VISUALIZZAZIONE DEL PERCORSO PIÙ BREVE

La funzionalità che vogliamo prendere in considerazione è quella di mostrare il percorso tra due posizioni con tanto di indicazioni stradali. Ecco il codice della funzione gestore del'evento di click sul tasto *"Mostra Percorso"* del Windows Form frmLatLong:

```
If (Not Me.m_locArrivo Is Nothing And Not
                   Me.m_locPartenza Is Nothing) Then
  Me.DrawPath(Me.m_locArrivo, Me.m_locPartenza)
Else
  MsgBox("Selezionare prima il punto di arrivo!",
                 MsgBoxStyle.OKOnly, "Attenzione!")
```

Per ottenere una così importante funzionalità basta richiamare il metodo *DrawPath(…)* che riceve la località di partenza e di arrivo come due oggetti di tipo Location appartenente alla stessa classe della Windows Form. Di seguito riportiamo il codice della procedura di calcolo del percorso:

```
Dim objLoc(2) As MapPoint.Location
  Dim oMap = Me.MapCtrl.ActiveMap
```

Pocket PC

Ricerca

Una funzionalità molto importante della nostra applicazione di posizionamento è ricerca di un luogo. La ricerca viene attivata specificando le informazioni della via, della Città e della provincia. Non tutte le informazioni possono essere inserite ma questo potrebbe portare ambiguità nella determinazione del luogo da ricercare. Questa ambiguità potrebbe nascere quando inseriamo la sola informazione della via; se, ad esempio, inserissimo solo "via Dei Mille" questo comporterebbe una lista di risultati compatibili con i criteri impostati. Possiamo risolvere questo problema invitando l'utente a selezionare il luogo da una lista di località compatibili con i criteri di ricerca impostati.

Pocket PC

```
oMap = Me.MapCtrl.ActiveMap
With oMap
    objLoc(1) = LocPartenza
    objLoc(2) = LocArrivo
End With
Try
    oMap.ActiveRoute.Waypoints.Add(objLoc(1), "start")
    oMap.ActiveRoute.Waypoints.Add(objLoc(2), "End")
    If Not oMap.ActiveRoute.IsCalculated Then
        oMap.ActiveRoute.Calculate()
       End If
       .........
   Catch e As System.Runtime.InteropServices.
                                        COMException
      .....
   End Try
....
```

Per effettuare il calcolo del percorso utilizziamo l'oggetto *Route* ricavato dall'istanza della mappa corrente *oMap* con l'invocazione della property *ActiveRoute*. La fase successiva impone l'aggiunta degli estermi del percorso utilizzando la *Collection Waypoints* dell'oggetto *Route*. Impostati gli estremi basta invocare il metodo *Calculate()* per la visualizzazione del percorso e della indicazioni stradali.

SOLUZIONE CON ASP.NET E MAPPOINT.NET

A titolo di esempio vediamo come poter sviluppare una semplice applicazione in ASP.NET che utilizza il WEB Service *MapPoint.NET*. Lo scopo di questa applicazione è solo quello di evidenziare una soluzione alternativa alla precedente. Inoltre, impareremo come implementare in modo molto originale la logica di renderering delle mappe su una pagina WEB. Apriamo un nuovo progetto ASP.NET in linguaggio C# . Chiameremo il progetto "GPSLocaltor_ServerWS". La pagina principale della WEB Application è *PrincipaleWF.aspx*. Nella finestra possiamo notare un WEB Control di tipo ListBox e il Button Vai. Il ListBox viene caricato con la descrizione dei veicoli censiti nel nostro database SQL Server. La logica che vogliamo implementare è quella di selezionare un veicolo e visualizzare la sua posizione e il percorso rispetto ad un luogo fissato. A tale scopo riportiamo il codice del gestore dell'evento Click sul tasto *"Vai"*:

```
string strUrl = "VisualizzaPosizione.aspx?descVeicolo=
                              "+ListVeicoli.SelectedItem.Text;
  Response.Redirect(strUrl);
```

dal codice si evince che viene chiamata la pagina *VisualizzaPosizione.aspx* passandovi come parametro la descrizione del veicolo selezionato. In Fig. 7 possiamo notare il risultato della visualizzazione della pagina di risposta. In modo particolare, osserviamo non solo la visualizzazione del percorso tra i due punti fissati ma

Coordinate

È possibile specificare le coordinate di longitudine e di latitudine, nel caso avessimo a disposizione queste informazioni, rendendo la ricerca più agevole. Una volta trovata la località di interesse, premendo il tasto "Mezzo più vicino" viene cercato il veicolo le cui ultime coordinate sincronizzate sono a distanza minima dal luogo della ricerca. La funzionalità interessante che MapPoint ci fornisce è quella di graficare sull'ActiveX il percorso tra i due luoghi e le indicazioni stradali.

Fig. 11: Visualizzazione del percorso e delle indicazioni stradali sulla pagina Web con MapPoint.NET.

anche le indicazioni stradali. La logica per ottenere questo risultato che ci permetterà di comprendere l'Object Model del WEB Service MapPoint.NET è di seguito risportata:

```
string param = Request.QueryString["descVeicolo"].ToString();
mapImage.ImageUrl = "MapPageForOnePoint.aspx?
                              descVeicolo="+param;
string str = (string)Session["indicazioniStradali"];
ScriviIndicazioniStradali();
```

Il codice precedente è stato inserito nel gestore *Page_Load* della WEB Form. Nella pagina *VisualizzaPosizione.aspx* è presente un WEB Control di tipo Image nel quale viene caricata l'immagine della mappa impostando la property *ImageURL*. Per ottenere la logica di rendering è stata progettata per lo scopo la pagina *MapPageForOnePoint.aspx* passando come parametro la descrizione del veicolo la cui posizione si vuole visualizzare. Prima di tutto vediamo cosa abbiamo inserito nel Gestore dell'evento di caricamento di questa ultima WEB Form (il codice lo trovate nel file *MapPagforonePoint.aspx.cs)*. Dal codice inserito possiamo notare come vengano istanziati i servizi di ricerca (*FindServiceSoap*), di rendering (*RenderServiceSoap*) e di routing (*RouteServiceSoap*) del WEB Service con i dati relativi all'account di accesso (parametri *Id* e *Psw*). Utilizziamo la classe di Accesso ai dati per recuperare le coordinate correnti del veicolo facendo una ricerca per descrizione del veicolo con il metodo *GetLatLongByDesc(..)*. Ottenute queste informazioni, in maniera analoga alla Windows Application precedente impostiamo gli estremi del percorso in un vettore che poi passiamo ad una istanza dell'oggetto *Route* per calcolare il percorso e le relative indicazioni stradali. In particolare, le indicazioni stradali vengono memorizzate in una variabile di Sessione per essere poi visualizzate dalla pagina chiamante. Le istruzioni successive costruiscono un vettore bidimensionale di oggetti *Pushpin* per l'inserimento dei dati da visualizzare sulla mappa.

Le ultime tre istruzioni permettono di ottenere un oggetto di tipo *MapImage* che viene utilizzato per modificare le informazioni sulle Intestazioni di pagina e scrivere i bit della immagine sul flusso di output.

Elmiro Tavolaro

Ottimizzazione

☑ Ottimizzazione ed elementi di programmazione avanzata.

Ottimizzare le Web Application Java

Con l'affermarsi della piattaforma Java come tecnologia leader per la creazione di applicazioni web, è sorta la necessità di spremere l'hardware a disposizione fino all'ultimo ciclo di clock per ottenere sempre le migliori prestazioni.

Java ha avuto in passato la nomea di linguaggio non eccelso in termini di prestazioni pure e sono state messe in atto, da parte di SUN e di altri produttori, diverse azioni per invertire questa condizione. D'altra parte, alcune importanti caratteristiche di Java lavorano contro le performance: i metodi virtuali, il controllo della sicurezza a runtime, l'indipendenza dalla piattaforma. Oggigiorno le Virtual Machine per la piattaforma Java hanno raggiunto livelli di sofisticazione che sono invidiati anche dai compilatori C++. Hotspot, ad esempio, esegue una ottimizzazione *dinamica* del codice mentre gira, oltre ovviamente a compilarlo nativamente ed a eseguire un inlining aggressivo del codice. I compilatori statici per il C++ si limitano invece a ottimizzare il codice *staticamente*, producendo un eseguibile immutabile.

OTTIMIZZARE

Ma anche il sistema più ottimizzato o il linguaggio più veloce possono essere utilizzati per produrre codice che gira lentamente, sprecando preziose risorse di processore o eccessiva memoria. In questo caso entrano in gioco la cultura performance dello sviluppatore che deve saper ottimizzare il codice quando serve. Questo vuol dire che, dopo il profiling dell'applicazione, una volta identificati i punti più spesso eseguiti, si dovrà procedere alla loro ottimizzazione. Infatti, una vecchia e nota regola dell'informatica spiega come l'80% del tempo macchina venga impiegato per eseguire il 20% del codice. Per questo motivo è sempre sconsigliata l'ottimizzazione *preventiva*, in quanto sono più i casi in cui questa strategia complica e rallenta inutilmente lo sviluppo rispetto a quelli in cui produce un risultato apprezzabile. Nonostante questo, è sempre bene tenere a mente alcuni tranelli classici da evitare a priori, nell'implementazione e nel design.

ALCUNE TECNICHE DI BASE

Il concetto alla base di quasi tutte le tecniche di ottimizzazione è quello di limitare alle sole operazioni essenziali le attività svolte dal processore. Alcuni accorgimenti per le performance classici nel mondo Java si basano proprio su questo concetto ed intervengono sull'utilizzo accorto degli elementi più basilari del linguaggio ed al loro impatto nella Virtual Machine.
Ad esempio:

Utilizzo di StringBuffer al posto dell'operatore di concatenamento "+". Come noto è possibile effettuare concatenamenti con l'operatore di addizione (+). Per ogni operazione di somma, però, il runtime di Java genera bytecode che crea oggetti di tipo *Stringa*, li concatena con *StringBuffer*, e poi li ritrasfroma in *Stringa*. Queste sono operazioni lente che possono essere ottimizzate utilizzando direttamente *StringBuffer* (Listato 1).
L'unico modo per utilizzare (+) senza overhead è su una singola riga di codice (Listato 2).

Utilizzare l'object pooling. Specialmente per le connessioni JDBC, che in ambito Web Application sono rilevanti ed hanno un costo di elaborazione considerevole, riutilizzare le connessioni al database consente di aumentare le prestazioni.

Listato 1

```
String s = "Uno";
s += "Due";
s += "Tre";
StringBuffer sb = new StringBuffer();
sb.append("Uno");
sb.append("Due");
sb.append("Tre");
```

Tecniche di ottimizzazione

Il concetto alla base di quasi tutte le tecniche di ottimizzazione è quello di limitare alle sole operazioni essenziali le attività svolte dal processore.

Ottimizzazione

```
Listato 2
String s = "Uno" + "Due" + "Tre";
```

In un contesto più prettamente Web, si possono identificare ulteriori semplici accorgimenti per le prestazioni:

Limitare l'utilizzo di System.out.println(). Spesso si utilizzano gli stream *out* ed *err* poiché è un metodo semplice per eseguire la tracciatura dell'applicazione, ad esempio a fini di debug. Questi stream sono solitamente raccolti dal Web Container e registrati all'interno di file di testo. L'intera operazione di raccolta è scrittura su file è abbastanza lenta, fino a sette volte più lenta con una chiamata per richiesta. Alcuni Web Container consentono di reindirizzare *out* ed *err* a "null" disabilitando l'effettivo output dei dati. Comunque è possibile considerare l'opportunità di condizionare l'effettiva stampa delle informazioni di log ad una variabile *final boolean* (Listato 3).

Escludere la sessione nelle JSP. Tramite la direttiva *<%@page session="false"%>* è possibile indicare al Web Container di non creare gli oggetti *HttpSession* realizzando un piccolo guadagno sugli overhead. Ovviamente questa direttiva può essere utilizzata solo per le pagine JSP che non richiedano l'uso delle informazioni presenti in sessione.

Direttive

Tramite la direttiva *<%@page session="false"%>* è possibile indicare al Web Container di non creare gli oggetti *HttpSession* realizzando un piccolo guadagno sugli overhead.

```
Listato 3
public class MyServlet extends HttpServlet {
private static final boolean debug = true;
public void service(HttpServletRequest req,
      HttpServletResponse resp) throws ServletException,
                                              IOException {
     if(debug) {System.out.println( req.toString() ); }
   }
}
```

INFORMAZIONI NELLE SESSIONI

Un discorso a parte merita la gestione delle informazioni in sessione. Le specifiche della tecnologia Servlet rendono facile l'utilizzo della sessione per memorizzare informazioni legate alle operazioni in corso. Il lato negativo di questa funzionalità è però che la sua gestione ha un notevole costo elaborativo ed alcune funzionalità dei Web Container più avanzati, come il *failover*, limitano l'uso che si può fare di questa caratteristica. La regola pratica più semplice da seguire è quella di memorizzare nelle sessioni pochi oggetti e molto semplici, ad esempio stringhe che rappresentano *ID* di oggetti più complessi, oppure oggetti *Home* di componenti EJB. L'utilizzo di molta memoria per la sessione dell'utente ha diversi risvolti negativi. Per prima cosa viene limitato il numero massimo di utenti che possono essere supportati dal singolo nodo (*Computer*). Ad esempio: se con una sessione di 10KB e 1000 utenti, la memoria utilizzata è di 9,7 MB, con sempre 1000 utenti, ma con una dimensione della sessione di 500KB, l'occupazione di memoria sarebbe di 488 MB. La memoria richiesta cresce dunque in modo lineare in funzione della quantità di informazioni memorizzate in sessione. In secondo luogo, è necessario considerare i tempi di accesso ai dati della sessione. Se l'applicazione è installata su un nodo singolo, la sessione è mantenuta in memoria e dunque i tempi di overhead per l'accesso a questi dati sono limitati. In questo caso, però, il sistema non ha caratteristiche di failover perché se dovesse cadere il singolo Web Container, l'applicazione sarebbe completamente inaccessibile da parte dell'utente. Per ovviare a questo problema si utilizzano *cluster* di nodi. Diversi Web Container vengono installati su altrettanti computer e messi in comunicazione tra di loro. Sebbene esistano diverse tecnologie utilizzate per implementare la comunicazione tra i vari nodi (e nessuno standard), il metodo più utilizzato (ad esempio da WebSphere) è quello di memorizzare i dati della sessione in un database comune. In questo modo tutti i nodi accedono allo stesso database ed hanno in comune i medesimi dati di sessione. In questo caso però, i tempi di recupero sono maggiori tanto maggiore è la dimensione della sessione. Inoltre, ci possono essere limiti fisici legati al database in uso ed al meccanismo di serializzazione. L'implementazione di WebSphere, ad esempio, serializza l'intera sessione tramite le API Java di serializzazione (*java.io.Serializable*) e scrive il risultato su un record del database. Se questo è un DB2 il limite è di 2MB, ma per altri database può essere diverso, in funzione della dimensione massima dei campi BLOB del database server.
Un'alternativa che offre WebSphere è la serializzazione di ciascuna chiave in sessione in altrettanti record del database, ma anche in questo caso il limite di ciascun elemento è di 2MB (per DB2). In altri casi, come ad esempio per WebLogic, la sincronizzazione delle sessioni avviene tramite cache condivise che vengono allineate da parte di ciascun nodo. Il problema è però che non esiste una garanzia al 100% che in qualsiasi momento le informazioni siano allineate, sopratutto se le informazioni da sincronizzare sono molte (richiedendo un overhead maggiore). Ne consegue che, per via del limite della memoria del computer o per quella del meccanismo di serializzazione, la dimensione della sessione deve essere contenuta. Per ottimizzare comunque i tempi di accesso ai dati della sessione in sistemi cluster, è possibile utilizzare un sistema personalizzato basato su JDBC, escludendo quello presente nel Web Container. Quest'ultimo, infatti, non può essere a conoscenza della logica dei dati presente in sessione che raramente sono sempre tutti necessari ad ogni pagina JSP. E' possibile dunque implementare una logica selettiva memorizzando in database le informazioni di sessione non in modo serializzato come fa il Web Container, ma in modo destrutturato, decomponendo le informazioni nei singoli elementi richiesti dalle singole pagine. Queste informazioni po-

tranno poi essere memorizzate nel database tramite JDBC. Quando richiesto, saranno recuperate dal database solo le informazioni strettamente necessarie all'operazione in corso.

UTILIZZO DI JDBC

Un approccio non corretto per quanto riguarda l'accesso alle connessioni JDBC, che però si utilizza quando magari il tempo a disposizione per sviluppare è poco, è quello di eseguire inutili connessioni al database. Quando però viene sollevato il problema delle performance, questo può essere un'area di intervento. Ad esempio, una chiamata a *DriverManager.getConnection*() all'interno del codice che viene eseguito in ogni chiamata, ad esempio nel metodo *service()* di una Servlet. Un esempio di errato approccio è presente nel listato 4.

Listato 4

```
public class MyServlet extends HttpServlet {
  //...
  public void service(HttpServletRequest req,
     HttpServletResponse resp) throws ServletException,
  IOException {
    //...
    Connection conn=DriverManager.getConnection(url);
    ResultSet rs = conn.executeQuery( query );
    //...
  }
}
```

Semplicemente eseguendo un caching della connessione è possibile ottenere un consistente risparmio di risorse, con tempi fino a tre volte inferiori. Un esempio è presente nel Listato 5.

Listato 5

```
public class MyServlet extends HttpServlet {
  //...
  javax.sql.DataSource ds = null;
public void service(HttpServletRequest req,
     HttpServletResponse resp) throws ServletException,
                                         IOException {
    //...
    Connection conn = ds.getConnection( url );
    ResultSet rs = conn.executeQuery( query );
    //...
  }
}
```

Nel listato 5 si presuppone che l'oggetto *DataSource* venga recuperato in un altro punto del codice. *DataSource* è una classe presente in JDBC 2.0, quindi è possibile utilizzarla solo in Web Container che dispongano di questo livello di specifiche. Inoltre, l'implementazione dei *DataSource* utilizzata, deve supportare il pooling delle connessioni JDBC all'interno del metodo *getConnection*(). Diversamente, il secondo blocco di codice non avrebbe vantaggi rispetto al precedente (Listato 4).

In alternativa è possibile implementare un pooling di oggetti *java.sql.Connection* con approcci tradizionali (tra l'altro il pooling di oggetti in Java è stato trattato in passato da ioProgrammo) od utilizzando package specifici disponibili open source su Internet, come ad esempio nel progetto Jakarta di Apache (*http://jakarta.apache.org*). Una buona implementazione del pooling delle connessioni al database può arrivare a ridurre i tempi fino ad un quarto. Se questo accorgimento viene poi utilizzato insieme al caching del DataSource stesso, i tempi possono essere ulteriormente ridotti. Il caching del DataSource presuppone il corretto posizionamento del codice di recupero di questo oggetto nel programma. Non è infatti consigliabile implementare il codice per l'ottenimento del DataSource ad ogni richiesta Web ma è più efficiente posizionarlo nel codice di inizializzazione del componente, ad esempio, nel caso di una Servlet, nel metodo *init()*. Solitamente, infatti, il DataSource viene recuperato dal Web Container tramite una chiamata JNDI. Eseguire questa chiamata al momento giusto concorre all'aumento delle performance e riduce i tempi fino a sei volte.

Listato 6

```
public class MyServlet extends HttpServlet {
  //...
  javax.sql.DataSource ds = null;
public void init(ServletConfig config)
  throws ServletException {
     //...
    Context ctx = new InitialContext();
    ds = (javax.sql.DataSource)ctx.lookup(
                                        "jdbc/SAMPLE");
    ctx.close();
    //...
  }
}
```

Nel listato 6 è presente un esempio di metodo *init()* che esegue la richiesta JNDI al Web Container per recuperare la risorsa SAMPLE sotto il gruppo jdbc. Per funzionare, ovviamente, nel Web Container deve essere definita questa risorsa JNDI e deve puntare ad un DataSource valida che punti al database in uso. Sempre in merito alle connessioni JDBC, è bene considerare quello che accade dopo che il programma ne ha fatto uso. La chiusura di Recordset e connessioni non è obbligatoria ed il programma in effetti gira anche senza queste chiamate. Il problema sorge con un uso intenso dell'applicazione. Essendo le risorse di connessioni finite, è possibile esaurirle e, sebbene i Web Container più evoluti abbiano meccanismi di recupero che liberano le connessioni dopo un timeout, questi tempi d'attesa potrebbero essere riscontrati dagli utenti dell'applicazione. Per evitare questi problemi è sufficiente ricordarsi di liberare le risorse quando non servono più. Un esempio è presente nel listato 7. Si noti che in questo esempio non è stato incluso il codice per intercettare le eccezioni *SQLException* che possono essere sollevate

Ottimizzazione

Pooling

Una buona implementazione del pooling delle connessioni al database può arrivare a ridurre i tempi fino ad un quarto.

Ottimizzazione

dal codice che opera su classi del package *java.sql*. Codice che dovrà essere presente in una implementazione completa della Servlet.

Listato 7
```
public class MyServlet extends HttpServlet {
  //...
public void service(HttpServletRequest req,
    HttpServletResponse resp ) throws ServletException,
                                              IOException {
    //...
    Connection conn = ds.getConnection( url );
    ResultSet rs = conn.executeQuery( query );
    //...
    rs.close();
    conn.close();
    //...
  }
}
```

THREADING E SINCRONIZZAZIONE

Sebbene l'architettura J2EE fornisca un ambiente già pronto e con servizi che si preoccupano di trattare con questioni complesse come la sincronizzazione ed il threading, è possibile identificare un paio di suggerimenti utili per velocizzare le applicazioni. Per prima cosa si consideri l'interfaccia *SingleThreadModel*. Le Servlet che implementano questa interfaccia dichiarano al Web Container che sarà questo a preoccuparsi dei problemi di rientranza del codice. La Servlet creata, di fatto, non supporterà chiamate multiple. L'approccio usuale dei Web Container è quello di risolvere il problema creando un oggetto Servlet per ogni richiesta (è il modo più semplice). In questo modo, però, potrebbe esserci un calo di prestazioni, se queste non sono gestite da un object pool. Questa informazione può essere recuperata dalla documentazione del Web Container in uso. Un approccio più prestante è dunque quello di evitare l'uso di *SingleThreadModel* (nel listato 8 è presente il codice da evitare) ma di implementare la sincronizzazione del codice con l'istruzione *synchronized*. Utilizzando questa parola chiave è necessario porre attenzione a minimizzare il codice sincronizzato. Nel listato 9 è presente un esempio sbagliato: tutto il corpo del metodo *service* è sincronizzato. In questo modo tutti gli accessi vengono serializzati per tutto il codice del metodo, cosa che forse può essere evitabile. Utilizzare la sincronizzazione in questo modo è quasi come se la servlet fosse *SingleThreadModel*.

J2EE

Sebbene l'architettura J2EE fornisca un ambiente già pronto e con servizi che si preoccupano di trattare con questioni complesse come la sincronizzazione ed il threading, è possibile identificare un paio di suggerimenti utili per velocizzare le applicazioni.

Listato 8
```
public class MyServlet extends HttpServlet
                      implements SingleThreadModel {
  //^^^ NON utilizzare SingleThreadModel !!!
  //...
public void service(HttpServletRequest req,
     HttpServletResponse resp) throws ServletException,
                                              IOException {
    //...
  }
}
```

Listato 9
```
public class MyServlet extends HttpServlet {
  //...
  int i = 0;
public void service(HttpServletRequest req,
    HttpServletResponse resp ) throws ServletException,
                                              IOException {
    //NON sincronizzare tutto il codice!!!
    synchronized( this ) {
      //...
      i++;
      //...
    }
  }
}
```

Un'alternativa è di sincronizzare solo il codice strettamente necessario, tipicamente quello che accede a variabili di classe della Servlet. Minore è il codice sincronizzato, minori sono i colli di bottiglia incontrati dalla Java Virtual Machine quando deve eseguire lo stesso codice da parte di più thread.

Listato 10
```
public class MyServlet extends HttpServlet {
  //...
  int i = 0;
public void service(HttpServletRequest req,
     HttpServletResponse resp) throws ServletException,
                                              IOException
  {
    synchronized( this )
    {
      i++;
    }
    //...
    //Maggior parte del codice
    //...
  }
}
```

CONCLUSIONI

Seguendo pochi e semplici accorgimenti è possibile spremere l'hardware a disposizione fino all'ultimo ciclo di clock per produrre applicazioni che abbiano prestazioni più elevate. Un sito veloce è gradevole ed utile per l'utente che lo visita, perché non deve perdere tempo ad attendere la risposta del server ma può visionare subito le informazioni di suo interesse. In questo modo l'utente fruisce di una experience piacevole che lo indurrà a tornare ancora sul vostro sito.

Massimiliano Bigatti

COM e .NET

Interazione tra COM e piattaforma .NET.

Utilizzare oggetti COM in applicazioni .NET

Il framework .NET fornisce funzionalità potenti e allo stesso tempo semplici da usare, per consentire l'interoperabilità fra nuovo codice managed e vecchio codice unmanaged, potenzialità necessarie fino quando la migrazione verso .NET non sarà avvenuta completamente.

Il framework Microsoft .NET fornisce gli strumenti e le potenzialità necessari ad interagire con codice non gestito, componenti COM, servizi COM+, type library esterne, oltre ai servizi messi a disposizione dal sistema operativo, ad esempio le API di Windows.

Naturalmente, in .NET molte cose sono variate rispetto al modello di codice non gestito. Cambiamenti, che riguardano i tipi, le eccezioni, gli errori e la loro gestione, l'invocazione dei metodi, l'esportazione delle interfacce e delle funzioni.

Il cosiddetto codice gestito è il codice che viene eseguito sotto il controllo del .NET runtime. Viceversa, il codice che gira al di fuori del *Common Language Runtime* (CLR), cioè il codice a cui eravamo abituati prima dell'arrivo di .NET, come ad esempio le API Win32 oppure i controlli ActiveX o i componenti COM, è detto codice unmanaged o non gestito.

Naturalmente, sono già stati scritti miliardi di righe di codice unmanaged e che potrebbero ancora fornirci un valido aiuto nell'arduo lavoro di sviluppatori, essendo oltretutto cresciuti con il comandamento che è inutile reinventare continuamente la ruota.

L'avvento della piattaforma .NET di Microsoft non ci costringerà quindi a rifare tutto da zero, e di tempo ne passerà prima che il codice gestito possa competere, in termini di presenza e quantità nei nostri sistemi, con il codice non gestito. Fino ad allora potremo continuare a mischiare codice managed con codice unmanaged e viceversa.

Ad esempio utilizzare componenti COM all'interno delle nostre applicazioni .NET potrebbe essere una buona idea.

In questo articolo vedremo come fare a scrivere un piccolo esempio in C#, linguaggio principe di .NET, in modo da poter utilizzare un componente COM.

UN SEMPLICE COMPONENTE COM

Cominciamo proprio dall'implementazione di un semplice ed immediato componente COM, la cui unica pretesa è quella di eseguire le classiche quattro operazioni di una calcolatrice e, volendo proprio esagerare, anche l'altrettanto classico calcolo del fattoriale e la funzione di memorizzazione e richiamo di un risultato di calcolo.

Per realizzare il nostro componente COM utilizzeremo Visual C++ 6, in modo da essere completamente scollegati dalla piattaforma .NET, dubbio che potrebbe sorgere a qualche lettore se utilizzassimo VC++ .NET, con il quale, è bene chiarirlo, è comunque possibile realizzare applicazioni o componenti unmanaged.

Apriamo dunque l'ambiente di sviluppo Visual C++ 6, e dal menu file scegliamo la voce *[New…]*, nel wizard cominciamo con lo scegliere la voce *[ATL COM AppWizard]*, e diamo un nome al nostro progetto, ad esempio *"CalcolatriceCOM"*. Nel passo successivo, selezioniamo l'opzione *"Dynamic Link Library (DLL)"* e premiamo *[Finish]*. A questo punto aggiungiamo un oggetto *ATL*, scegliendo dal menù *[Insert]* la voce *[New ATL object…]*. Nella finestra di dialogo selezioniamo *[Simple Object]* e diamo un nome al nostro oggetto, ad esempio *Calc*. L'ambiente creerà per noi la classe *CCalc* che implementa l'interfaccia *ICalc*.

Adesso non resta che aggiungere i metodi che eseguono le operazioni della calcolatrice.

Nella dialog box che contiene il *workspace*, basta cliccare con il tasto destro sul nome *ICalc*, e scegliere la voce *[Add. Method…]*, mentre nella successiva finestra di dialogo (vedi Fig. 1), inseriremo il nome del metodo e i suoi parametri, specificando anche se si trattano di parametri di ingresso oppure di uscita, ad esempio per il metodo che esegue la differenza, possiamo scrivere nella casella *Parameters*:

La classe TypeLibConverter

La classe *TypeLibConverter* (contenuta nel namespace *System.Runtime.InteropServices namespace*) fornisce i metodi per convertire *coclass* e *interfacce COM*, contenute in una *type library*, in metadati, e viceversa una libreria di tipi COM in un assembly gestito.

COM e .NET

Fig. 1: Il Wizard che ci aiuta nella costruzione di un metodo.

```
[in] double d1, [in] double d2, [out,retval] double* dDiff
```

[out,retval] indica che il parametro *dDiff* è un argomento di ritorno del metodo, dunque deve anche essere necessariamente un puntatore.
Ripetiamo gli stessi passi per tutte le operazioni che vogliamo implementare, mentre per la funzione di memorizzazione possiamo aggiungere alla nostra interfaccia una *Property*, dandole ad esempio il nome *Mem*, ed ancora *double* come tipo.
Ci penserà Visual C++ a scrivere due metodi *put_Mem* e *get_Mem*, rispettivamente per memorizzare e per richiamare il valore.
Finora abbiamo però solo un'interfaccia, non ci resta che implementarla nella classe *CCalc*.
Aggiungiamo a quest'ultima una variabile membro che funge da memoria, nel quale scriveremo il valore *double* che vogliamo memorizzare, e dalla quale potremo anche richiamarlo.
L'implementazione dei metodi è semplicissima, data l'immediatezza dell'esempio, e la tralasciamo dato che lo scopo dell'articolo non è questo. Naturalmente sul CD è presente l'esempio completo di sorgente e della libreria già compilata.

Early & Late Binding

Prima che un client possa chiamare i metodi di un server COM, è necessario ottenere l'indirizzo in memoria dei metodi stessi, cioè effettuare il binding. Se tale risoluzione avviene a compile-time si parla di early binding. Con il late binding, invece, l'associazione avviene a runtime, quando viene esplorato il componente alla ricerca dei metodi.

UTILIZZARE COM IN C#

Una volta compilato il progetto *ATL COM* di visual C++, otterremo un file con estensione *tlb (type library)*, contenente appunto le definizioni dei tipi. Ma come facciamo ad usarle in ambiente managed? Spesso i produttori di componenti COM forniscono anche degli assemblies già pronti all'uso, chiamati *Primari Interop Assemblies*, ma se così non fosse dobbiamo fare da soli questa operazione di import, che in realtà si traduce nella generazione di un assembly wrapper.
Per generare questa libreria wrapper, a partire da un file *tlb* o dalla *dll*, abbiamo diverse possibilità equivalenti. Ci limiteremo alle due più semplici e probabilmente più comuni.
Se abbiamo a disposizione Visual Studio .NET, l'operazione da eseguire è quella di aggiungere al progetto un riferimento alla type library.
Una volta aperto l'ambiente di sviluppo e creato un nuovo progetto C#, nel nostro caso un'applicazione Windows, basta cliccare nel menù *Project* (o *Progetto* per la versione italiana) la voce *References* (*Aggiungi riferimento…*), e passare alla scheda COM. Selezioniamo dall'elenco la libreria se essa è già registrata nel sistema oppure sfogliamo per cercare il file tlb (Fig. 2).

Fig. 2: L'aggiunta di un componente COM in VS.NET.

Visual Studio .NET genera automaticamente l'assembly wrapper, come è facile verificare nei riferimenti visualizzati nella finestra *Esplora Soluzioni* (Fig. 3), o direttamente nella directory di output del progetto, dove verrà generato un file del tipo *interop.CalcolatriceCOMLib.dll*.

Fig. 3: Ecco pronto il nostro componente.

TYPE LIBRARY IMPORTER

L'alternativa a Visual Studio .NET è quella di usare un'utility a linea di commando, *tlbimp.exe*, che converte le classi e le interfacce contenute in una type library

Fig. 4: Schema di un componente COM importato in .NET.

COM in metadata, creando un proxy assembly, detto *Runtime Callable Wrapper* (RCW), il cui compito è quello di gestire le chiamate ai metodi del componente e di restituire valori di ritorno compatibili con il *Common Language Runtime* (Fig. 4).
Per generare un interop assembly con *tlbimp.exe* basta lanciare il comando seguito dal nome del file tlb o della dll del componente COM. Ad esempio, se abbiamo ottenuto dal nostro progetto il file *CalcolatriceCOM.tlb*, il comando

```
tlbimp CalcolatriceCOM.tlb /verbose
```

produrrà una dll *CALCOLATRICECOMLib.dll* che potremo utilizzare come un qualunque assembly .NET (Fig. 5).

Fig. 5: **Il processo di wrapping è andato a buon fine.**

Con entrambi i metodi esposti, i metadati saranno inseriti in un *namespace* con lo stesso nome della type library da cui sono stati generati. Nel nostro esempio, avendo convertito la libreria *CalcolatriceCOM* in un file assembly *CalcolatriceCOMLib.dll*, troveremo la classe *CalcolatriceCOMLib.CalcClass*.
Il namespace generato può comunque essere personalizzato utilizzando il tool *tlbimp* con l'opzione */namespace*: seguita dal nome desiderato.

ISPEZIONE CON ILDASM

Avendo progettato e scritto noi il componente COM, conosciamo già i tipi contenuti nella libreria, e quindi nell'assembly, ma come facciamo se non abbiamo nessuna informazione sull'assembly?
L'utility *ILDASM* è uno strumento che consente di analizzare, in maniera grafica, i metadati contenuti in qualunque file eseguibile *PE (Portable Executable)* managed, comprese le librerie importate da componenti COM.
Se ad esempio, dopo aver lanciato l'utility, apriamo il file *CALCOLATRICECOMLib.dll*, otterremo un albero delle definizioni dei tipi e la firma dei metodi contenuti nell'assembly wrapper (Fig. 6), oltre a poter generare il dump in formato *IL (intermediate language)*.

Fig. 6: **L'albero rappresentativo della nostra classe.**

UTILIZZO DEI METODI

Un client .NET può chiamare i metodi di un oggetto COM, settare le sue proprietà, e gestire gli errori e gli eventi generati dal server COM. Vediamo come farlo utilizzando il componente *CalcolatriceCOM* nel progetto C#, per mezzo dell'assembly wrapper *CALCOLATRICECOMLib*.
Creiamo innanzitutto una minimale interfaccia grafica, costituita da una form, con due caselle di testo per inserire gli operandi, una per il risultato, ed i pulsanti per gestire le operazioni della calcolatrice (Fig. 7).

Fig. 7: **L'interfaccia della nostra applicazione.**

Aggiungiamo alla classe C# una variabile membro *m_calc* di tipo *CalcClass*, che inizializzeremo nel costruttore, come un qualsiasi oggetto:

```
m_calc=new CALCOLATRICECOMLib.CalcClass();
```

Chiamare i metodi dell'oggetto *m_calc* non è per nulla differente da come siamo abituati a fare, sia in ambito managed che unmanaged.
Non dobbiamo nemmeno preoccuparci del tipo dei parametri, dato che il marshalling avverrà in maniera automatica, ad esempio un tipo *System.String* in C#, verrà convertito in *BSTR*.
Per quanto riguarda invece i parametri che abbiamo contrassegnato con la dicitura *[out,retval]*, verranno restituiti come valori di ritorno dai metodi, infatti il processo di conversione li rimuove automaticamente dalla signature.
Gli errori che avvengono all'interno del componente COM, saranno anch'essi tradotti in eccezioni nell'ambito del CLR.
Abbiamo quindi a disposizione, per la calcolatrice, tutti i metodi esposti dal componente scritto in C++, e non dobbiamo far altro che gestire i click sui pulsanti, leg-

COM e .NET

Garbage Collection

Utilizzando componenti com all'interno del framework .NET, si continua a beneficiare dell'intervento del Garbage Collector. Come noto, la cancellazione degli oggetti da parte di .NET avviene su base non deterministica quindi, nel caso in cui i componenti COM utilizzino risorse in modo particolarmente dispendiose, sarà bene prevedere un metodo Dispose() che permetta di rilasciare esplicitamente l'oggetto.

COM e .NET

gere gli operandi dalle caselle di testo e scrivere il risultato nella casella di testo in basso.
Scriviamo un metodo che gestisca le quattro operazioni aritmetiche, distinguendo l'operazione da effettuare con uno *switch*:

```
/// <summary>
/// Esegue l'operazione aritmetica specificata
/// a scelta tra "+","*","-","/"
/// </summary>
/// <param name="op">operazione da
                                    eseguire</param>
/// <returns>il risultato dell'operazione</returns>
private double ExecuteOp(string op)
{
   double d1;
   double d2;
   try
   {
      d1=Double.Parse(textBox1.Text);
      d2=Double.Parse(textBox2.Text);
   }
   catch(FormatException)
   {
      MessageBox.Show("Inserire valori numerici");
      return 0.0;
   }
   switch(op)
   {
      case "+":
      return m_calc.Sum(d1,d2);
      case "-":
      return m_calc.Diff(d1,d2);
      case "/":
      return m_calc.Div(d1,d2);
      case "*":
      return m_calc.Mult(d1,d2);
      default:
      MessageBox.Show("Operazione non supportata");
      return 0.0;
   }
}
```

Come vedete, una volta decisa l'operazione e convertiti i valori inseriti nelle *TextBox* in valori *double* (a meno di una eccezione *FormatException* eventualmente gestita), basta richiamare i metodi corrispondenti offerti dal componente COM.
Ad esempio, il seguente è il metodo che gestisce il click sul pulsante "+":

```
private void btSum_Click(object sender,
                                 System.EventArgs e)
{
   m_txtRisultato.Text=ExecuteOp("+").ToString();
}
```

Analogamente, per calcolare il fattoriale di un numero intero, basta scrivere un metodo come il seguente:

```
try
{
   int n=int.Parse(textBox1.Text);
   m_txtRisultato.Text=m_calc.Fattoriale(n).ToString();
}
catch(FormatException)
{
   MessageBox.Show("Inserire un valore numerico");
}
```

GETTING E SETTING DELLE PROPRIETÀ

Le interfacce COM possono anche includere proprietà: nella nostra libreria ne abbiamo implementato una che gestisce la memoria della calcolatrice. Le operazioni di *Get* e *Set* vengono effettuate in maniera perfettamente tradizionale. Il seguente codice mostra come settare la proprietà *Mem* e come ricavarne il valore:

```
//Setting
m_calc.Mem=Double.Parse(m_txtRisultato.Text);

//Getting
textBox1.Text=m_calc.Mem.ToString();
```

GESTIONE DEGLI ERRORI

Gli eventuali errori o le eccezioni lanciate all'interno del componente COM possono essere gestite da un client .NET nella medesima maniera in cui si gestiscono le eccezioni o gli errori managed. Vediamo direttamente un esempio.
Nel nostro componente COM *Calcolatrice* modifichiamo il metodo *Div* in modo che il metodo fallisca se tentiamo una divisione per zero. Vi chiederete: "ma che modifiche bisogna fare? Una divisione per zero non darebbe comunque un errore?". Beh, è vero se trattiamo divisione fra tipi interi e decimali, ma non lo è dividendo un double per zero, visto che la divisione fra double in C# viene eseguita secondo le regole IEEE 754, e quindi otterremo un valore + o – infinito. Per far fallire il metodo *Div* dobbiamo allora esplicitamente controllare che il valore del secondo operando non sia zero, e sono in questo caso eseguire la divisione:

```
STDMETHODIMP CCalc::Div(double d1, double d2,
                                    double *dQuoz)
{
   if(d2==0) return E_FAIL;//se d2 è null oil metodo fallisce
      *dQuoz=d1/d2;
   return S_OK;
}
```

In questa maniera, il codice C#, tentando di eseguire una divisione per zero, otterrà il fallimento del metodo, che verrà tradotto in una eccezione di classe *System.Runtime.InteropServices.COMException*, non bisogna allora fare altro che gestirla con un

Sul Web

Documentazione on-line MSDN,
msdn.microsoft.com

www.gotdotnet.com

classico blocco *try/catch*:

```
private void btDiv_Click(object sender, System.EventArgs e)
  {
    try
    {
      m_txtRisultato.Text=ExecuteOp("/").ToString();
    }
    catch(System.Runtime.InteropServices.COMException)
    {
      MessageBox.Show("Divisione per zero!");
    }
}
```

LATE BINDING

Utilizzando metodi di reflection possiamo generare a Runtime un oggetto *CalcClass* e chiamarne i metodi, evitando anche di utilizzare i riferimenti alla libreria COM. Vediamo ad esempio come fare ad implementare il metodo che calcola la differenza con tale metodologia, detta anche late binding (vedi box laterale).
Innanzitutto, è necessario creare un tipo *CalcolatriceCOM.Calc*, dal quale poi generare l'oggetto *CalcClass* invocando il metodo statico *CreateInstance* della classe *Activator*:

```
Type calcType=Type.GetTypeFromProgID(
                              "CalcolatriceCOM.Calc");
  object tCalc=Activator.CreateInstance(calcType);
```

A questo punto bisogna preparare un array degli argomenti ed invocare il metodo *Diff*, dell'oggetto creato a runtime, specificando come secondo parametro il flag *InvokeMethod*, ad indicare proprio che si sta invocando un metodo:

```
  object[] args={Double.Parse(textBox1.Text),
                          Double.Parse(textBox2.Text)};
  double dDiff=(double)calcType.InvokeMember("Diff",
                            BindingFlags.InvokeMethod,
                                    null, tCalc, args);
  m_txtRisultato.Text=dDiff.ToString();
```

Il *late binding* può essere l'unica soluzione se non si ha a disposizione un assembly wrapper, ma le prestazioni sono certamente penalizzate, rispetto al-l'*early binding*, oltre ad una serie di fattori che non possiamo valutare con certezza fino all'esecuzione, ad esempio non è detto che la risoluzione del tipo COM avvenga correttamente, o che l'invocazione del metodo sia corretta, o che gli argomenti siano adeguati.

RILASCIARE IL COMPONENTE COM

Sebbene il *Runtime Callable Wrapper* sia gestito dal *Common Language Runtime*, e dunque sottoposto prima o poi all'azione di garbage collection, è possibile liberare esplicitamente le risorse utilizzate da un oggetto COM, rimuovendolo dalla memoria con una chiamata diretta al metodo statico *ReleaseComObject* della classe *Marshal*, che fa parte del namespace *System.Runtime.InteropServices*.
Nel nostro caso possiamo ad esempio inserire la chiamata seguente nel metodo *Dispose()* della form principale, in modo che alla chiusura dell'applicazione avvenga il rilascio dell'oggetto *m_calc*:

```
Marshal.ReleaseComObject(m_calc);
```

COMPILARE E DISTRIBUIRE IL PROGETTO

Utilizzando l'ambiente Visual Studio.NET non c'è nient'altro da fare, se non lanciare la generazione del progetto, ed eseguire il file eseguibile ottenuto, non dimenticando che, distribuendo la nostra applicazione, bisogna anche includere l'assembly wrapper generato, senza il quale verrà generata un'eccezione *FileNotFoundException*, e naturalmente la dll COM vera e propria, che deve anche essere registrata nel sistema, ad esempio con il comando:

```
regsvr32 CalcolatriceCOM.dll
```

In caso contrario l'applicazione genererà, appena tenteremo di eseguirla, un'eccezione *COMException*. Se invece non possiamo permetterci Visual Studio .NET o vogliamo semplicemente lavorare con il compilatore a linea di comando *csc*, dobbiamo necessariamente utilizzarlo con l'opzione */reference*, specificando il percorso della libreria di tipi COM:

```
csc nomefile.cs /reference:CalcolatriceCOMLib.dll
```

Abbiamo a questo punto terminato il progetto e possiamo allora eseguire l'applicazione ed utilizzarla, semplicemente con un doppio click sull'eseguibile o lanciando il comando dal prompt.

CONCLUSIONI

.NET, nelle intenzioni di molti, è il futuro, ma negli anni passati generazioni di programmatori hanno sviluppato e messo in circolazione una miriade di componenti COM e controlli ActiveX, è fondamentale quindi, prima di voltare definitivamente pagina, poter riusare quanto già sviluppato, e magari anche il contrario, scrivere componenti .NET da usare in qualche vecchio applicativo.
Questo articolo ha mostrato come utilizzare un componente COM in un applicativo C#, in attesa della migrazione completa.

Antonio Pelleriti

Bibliografia

• **PROGRAMMARE VISUAL C++ – V EDIZ.** ***Kruglinski, Sheperd, Wingo*** **(Mondadori Informatica)**

• **PROFESSIONAL C# II EDIZ.** ***Robinson*** **(Wrox Press)**

• **THINKING IN C#** ***B. Eckel, L. O'Brien***

• **PROGRAMMING C#** ***Jesse Liberty*** **(O'Reilly)**

INBox L'esperto risponde...

.NET Assembly

Cara ioProgrammo, vorrei avere delle delucidazioni in merito al .Net Framework di Microsoft. Da qualche tempo sto studiando questa nuova piattaforma perché vorrei aggiornare le mie conoscenze e passare dalla programmazione in Visual Basic alla programmazione in .Net. Vorrei sapere qualcosa in più sui cosiddetti assembly. Ho trovato diverse, e discordanti, definizioni... ne vorrei una definitiva! Vi ringrazio per l'attenzione e per il vostro magnifico lavoro.

Gaetano Parise

In .Net il concetto di assembly include sia file .EXE sia file .DLL e può dunque essere o una applicazione vera e propria o una libreria. Un assembly può essere costituito da uno o più file e rappresenta un gruppo di risorse, definizioni ed implementazioni di tipi, è inoltre possibile che un assembly contenga riferimenti ad altri assembly. L'elenco delle risorse che costituiscono l'assembly è contenuto in una porzione di dati chiamata "*manifesto*". Il manifesto è parte integrante dell'assembly, cosa che permette all'assembly di "auto-descriversi". Questa caratteristica è fondamentale per la gestione delle versioni e per la distribuzione delle applicazioni. In ogni assembly esistono metadati che consentono di descrivere pienamente i tipi di dato contenuti nel componente e la struttura che li tiene assieme.

Astrazioni

Gentile Redazione, evito i complimenti di rito, sottolineando che il mio apprezzamento nei vostri confronti è riscontrabile dal fatto che rinnovo l'abbonamento alla vostra rivista da tre anni. Ho deciso di scrivervi perché vorrei dei chiarimenti sul concetto di classe astratta e suo utilizzo. Grazie e continuate così!

Giulio

Una classe astratta è una classe "incompleta". Essa raggruppa un insieme di variabili e metodi, ma alcuni dei suoi metodi non contengono istruzioni, essi dovranno essere definiti in una classe ereditata da questa classe astratta. In generale serve a definire, a grandi linee, il comportamento di una classe di oggetti senza forzare l'implementazione dei dettagli dell'algoritmo. Si può pensare ad una classe *CatenaDiMontaggio* che, a partire da dei componenti base, realizzi una automobile. Ci sarà bisogno di classi più specifiche, derivate dalla classe base, che specifichino più dettagliatamente la costruzione delle varie parti di un'automobile al fine di costruire dei modelli concreti e differenziati. Si potranno avere *CatenaDiMontaggio_Fiat_Punto*, *CatenaDiMontaggio_Fiat_ Stilo* o, per i più fortunati, *CatenaDiMontaggio_ Ferrari_Maranello*, anche se dubitiamo fortemente che quest'ultima possa avere molti punti in comune con le prime due. Un esempio "più informatico" può essere una classe *VettoreOrdinato*. Potremmo pensare di costruire, magari in Java, una classe che accolga in un array oggetti di qualsiasi tipo e che preveda un metodo *ordinaVettore* che si occupi appunto di riordinare gli oggetti. È chiaro che il confronto tra gli oggetti sarà possibile solo a patto di definire il tipo di oggetto. Dichiareremo dunque astratto il metodo *ordinaVettore*, e di conseguenza la anche la classe *VettoreOrdinato* risulterà astratta. Alle sottoclassi di *VettoreOrdinato* è lasciato l'onere di implementare il metodo. Un ultimo chiarimento. Ovviamente non è possibile istanziare un oggetto a partire da una classe astratta: così come sarebbe difficile mettere in piedi una fabbrica di automobili senza specificare il modello, sarebbe altrettanto arduo istanziare un oggetto da *VettoreOrdinato*, così come lo abbiamo definito in precedenza.

Valore e riferimento

So di chiedervi una cosa un po' trita, ma sono alle prese con il C++ e vorrei avere qualche delucidazione sulla differenza fra passaggio per valore e passaggio per riferimento. Un grazie e mille complimenti per la qualità del vostro lavoro.

Gianluca

Passare ad una funzione un parametro per valore, equivale a passare una copia del contenuto di una variabile. Il passaggio per valore consente di mantenere inalterata la variabile utilizzata nel codice chiamante. La funzione invocata potrà dunque operare sul valore, appunto, della variabile senza andare a toccare il contenuto di variabili esterne ad esso. Il passaggio per valore è molto utilizzato e garantisce una maggiore indipendenza fra le varie parti di codice. Il passaggio per indirizzo permette di specificare un indirizzo di memoria su cui agire, consentendo alla funzione chiamata di modificare le variabili dichiarate nel codice chiamante:

```
#include <iostream.h>
int somma(int* add1, int* add2); //prototipo
main()
{
  int a = 7;
  int b = 13;
  int c;
  c = somma(&a, &b);
  cout << a << " + " << b << " = " c << endl;
}
```

Del tutto simile al passaggio per indirizzo è il passaggio per riferimento, che garantisce una maggiore semplicità quando si invoca una funzione. Nel richiamare la funzione non sarà necessario specificare l'operatore di indirizzamento:

```
#include <iostream.h>
int somma(int& add1, int& add2); // prototipo
main()
{
  int a = 7;
  int b = 13;
  int c;
  c = somma(a, b);
  cout << a << " + " << b << " = " c << endl;
}
```

Biblioteca

ON LINE

GameDev.NET

News, articoli, risorse, forum, community per tutti gli sviluppatori di videogame. Uno dei migliori siti in assoluto per l'argomento programmazione giochi.

http://www.gamedev.net/

CShrp.NET

Un grande portale sul linguaggio C#, sono presenti diversi articoli tecnici, codice sorgente pronto al download, news, interviste a "personaggi" del mondo della programmazione, news e quant'altro possa essere d'aiuto allo sviluppatore C#.

http://www.cshrp.net/

DotNetExtreme

Un sito dedicato a tutti coloro che vogliono iniziare a sviluppare applicazioni basate sulla nuova piattaforma Microsoft .NET. Centinaia di articoli tecnici dedicati a VB.NET, ASP.NET, C#, ADO.NET, JScript.NET, Mobile.NET, SOAP/XML, WebServices.

http://www.dotnetextreme.com/

Lezioni di C++

Un testo che rende particolarmente agevole l'apprendimento dei fondamenti della programmazione e del codice C++.
L'autore John Smiley, presidente della Smiley and Associates, un'azienda di consulenza informatica, è autore di altri otto libri e, con Lezioni di C++, ha pensato bene di fornire un testo adatto ad un lettore neofita, che non ha nessuna esperienza in campo di programmazione e che comunque si appresta a voler apprendere un linguaggio come il C++ che, sicuramente, non si annovera tra i linguaggi più semplici nell'apprendimento.
Gli argomenti, secondo lo stile dell'autore e ormai divenuto un suo standard, sono trattati in modo semplice e "divertente".
Tra gli argomenti trattati: imparare i concetti della programmazione applicabili a più linguaggi; sviluppare delle competenze in C++ per il mondo reale e utilizzare la programmazione a oggetti; lavorare con le variabili, le costanti e i dati tipizzati del C++.

Difficoltà: Medio – Alta • Autore: J.Smiley • Editore: McGraw-Hill
http://www.informatica.mcgraw-hill.it • ISBN: 88-386-4325-3 • Anno di pubblicazione: 2003
Lingua: Italiano • Pagine: 573 • Prezzo: € 32,00

Hacker Pronto Intervento

Il testo fornisce un approccio professionale alle indagini sugli attacchi informatici; gli esempi che accompagnano la lettura del testo, sono dettagliati di tecniche di rinvenimento e di analisi del materiale più significativo, il tutto applicato a episodi realmente accaduti.
Grazie alla lettura di questo testo, si sarà in grado di carpire le diverse metodologie che conducono alla risoluzione di un attacco da parte dei famigerati Hacker, sono ben spiegate sia le tecniche utilizzate per portare a termine l'attacco, sia i metodi, le applicazioni, i trucchi per evitarli.

Difficoltà: Media • Autore: K.Mandia, C.Prosise • Editore: Apogeo http://www.apogeonline.com
ISBN: 88-503-2026-4 • Anno di pubblicazione: 2003 • Lingua: Italiano • Pagine: 438
Prezzo: € 35,00

VB.NET Programmazione dei database

Il rilascio da parte di Microsoft della piattaforma .NET, ha dato un forte impulso alla "rivoluzione" della programmazione, nuovi concetti da imparare e applicare, nuove metodologie; l'ambiente .NET di Microsoft offre diverse novità anche per chi si accinge a creare applicazioni per database; nella fattispecie, Visual Basic.NET, risulta fortemente potenziato, per quanto riguarda la gestione degli oggetti ADO, l'integrazione con XML, l'uso di transazioni e delle Stored Procedure.
Il volume in oggetto tratta questi e tanti altri argomenti con un occhio di riguardo al "vecchio" programmatore Visual Basic, proprio per dar conto delle notevoli diversità di approccio nella costruzione di applicazioni per la nuova piattaforma.

Difficoltà: Media • Autore: G.Naccarato, G.Malorgio • Editore: Apogeo
http://www.apogeonline.com • ISBN: 88-503-2051-5 • Anno di pubblicazione: 2003
Lingua: Italiano • Pagine: 382 • Prezzo: € 30,00